robotron

DCP

SOFTWARE
Dokumentation

Anleitung für den Assemblerprogrammierer

Teil I – Heft 1

Die vorliegende 1. Auflage der Dokumentation "Anleitung fuer den Assemblerprogrammierer" unter DCP 3.2 entspricht dem Stand vom 30.6.87 und unterliegt nicht dem Aenderungsdienst.

Nachdruck, jegliche Vervielfaeltigung oder Auszuege daraus sind unzulaessig.

Die Dokumentation wurde durch ein Kollektiv des

VEB Robotron Buchungsmaschinenwerk Karl-Marx-Stadt

erarbeitet.

Bitte senden Sie uns Ihre Hinweise, Kritiken, Wuensche oder Forderungen zur Dokumentation zu.

VEB Robotron Buchungsmaschinenwerk
Karl-Marx-Stadt
PSF 129
Karl-Marx-Stadt
9010

Die "Anleitung fuer den Assemblerprogrammierer" besteht aus zwei Teilen

Teil 1 enthaelt:

I.	CPU - Befehlsbeschreibung	Heft 1
II.	Assembler (MASM)	Heft 2

Teil 2 enthaelt:

III.	Editoren (EDLIN, BE)	Heft 3
IV.	Bibliotheksverwalter (LIB)	Heft 3
V.	Binder (LINK)	Heft 4
VI.	Debugger (SYMDEB)	Heft 4
VII.	MAKE	Heft 4

I N H A L T S V E R Z E I C H N I S

		Seite
I.	CPU - BEFEHLSBESCHREIBUNG	6
1.	Programmiermodell und Registerarchitektur des K1810 WM86	6
1. 1.	Allgemeines	6
1. 2.	Registerstruktur	6
1. 3.	Speicheradressen, Segmente	8
1. 4.	Die Register des K1810 WM86	10
1. 4. 1.	Hauptregister/allgemeine Register	10
1. 4. 2.	Zeigerregister	11
1. 4. 3.	Indexregister	11
1. 4. 4.	Befehlszeiger	12
1. 4. 5.	Segmentregister	12
1. 4. 6.	Flagregister	13
2.	Metasprache	14
2. 1.	Allgemeine Elemente	14
2. 2.	Operandenelemente	15
2. 3.	Flageinstellung	16
3.	Adressierung	16
3. 1.	Adressmodi	16
3. 1. 1.	Registeradressierung	17
3. 1. 2.	Adressierung ueber Direktwert	17
3. 1. 3.	Adressierung ueber Direktadresse	17
3. 1. 4.	Register-indirekt-Modus	18
3. 1. 5.	Basisadressmodus	18
3. 1. 6.	Indexadressmodus	18
3. 1. 7.	Basis- und Indexadressmodus kombiniert	19
3. 2.	Befehlsaufbau	19
4.	Beschreibung der Befehle	21
4. 1.	Praefixe	21
4. 1. 1.	Wiederholungspraefix	21
4. 1. 2.	Bus-Lock-Signal fuer folgenden Befehl aktivieren	22
4. 2.	Ladebefehle	23
4. 2. 1.	Uebertragen von Daten	23
4. 2. 2.	Laden Datensegmentregister	26
4. 2. 3.	Laden Extrasegmentregister	26
4. 2. 4.	Laden effektive Adresse	28
4. 3.	Austauschbefehle	29
4. 4.	Stackbefehle	30
4. 4. 1.	Stack leeren	30
4. 4. 2.	Flags aus dem Stack laden	32
4. 4. 3.	Stack fuellen	33
4. 4. 4.	Flags in den Stack laden	34
4. 5.	Schleifenbefehle	35
4. 5. 1.	Schleifenbefehl	35
4. 5. 2.	Bedingte Schleifenbefehle mit Sprung bei Gleich oder Null	36

		Seite
4. 5. 3.	Bedingte Schleifenbefehle mit Sprung bei Ungleich oder Nichtnull	37
4. 6.	Arithmetikbefehle	38
4. 6. 1.	Addition ohne Uebertrag	38
4. 6. 2.	Addition mit Uebertrag	40
4. 6. 3.	Dekrementieren	41
4. 6. 4.	Inkrementieren	43
4. 6. 5.	Subtraktion ohne Uebertrag	44
4. 6. 6.	Subtraktion mit Uebertrag	46
4. 6. 7.	Multiplikation vorzeichenlos	48
4. 6. 8.	Multiplikation vorzeichenbehaftet, ganzzahlig	49
4. 6. 9.	Division vorzeichenlos	50
4. 6.10.	Division vorzeichenbehaftet, ganzzahlig	52
4. 6.11.	Leeroperation	53
4. 6.12.	Arithmetische Negation - Zweierkomplement	54
4. 6.13.	Umkodierung von AL	56
4. 6.14.	Vergleich	58
4. 7.	Logikbefehle	58
4. 7. 1.	Logisches UND	60
4. 7. 2.	Logisches ODER	60
4. 7. 3.	Exklusiv-ODER	62
4. 7. 4.	Logisches UND ohne Resultatsabspeicherung	64
4. 7. 5.	Logisches Komplement	66
4. 8.	Sprungbefehle	67
4. 8. 1.	Bedingte Sprungbefehle	67
4. 8. 2.	Unbedingter Sprung	69
4. 9.	Verschiebe- und Rotationsbefehle	69
4. 9. 1.	Verschiebebefehle	71
4. 9. 1. 1.	Arithmetische und logische Linksverschiebung	71
4. 9. 1. 2.	Arithmetische Rechtsverschiebung	73
4. 9. 1. 3.	Logische Rechtsverschiebung	74
4. 9. 2.	Rotationsbefehle	76
4. 9. 2. 1.	Linksrotation	76
4. 9. 2. 2.	Rechtsrotation	77
4. 9. 2. 3.	Linksrotation durch CF	79
4. 9. 2. 4.	Rechtsrotation durch CF	81
4.10.	Flagbefehle	82
4.10. 1.	Uebertragsflag setzen	82
4.10. 2.	Uebertragsflag ruecksetzen	83
4.10. 3.	Richtungsflag setzen	83
4.10. 4.	Richtungsflag ruecksetzen	84
4.10. 5.	Interruptflag setzen	84
4.10. 6.	Interruptflag ruecksetzen	85
4.10. 7.	Flags in das Register AH laden	85
4.10. 8.	Flags von AH in das Flagregister laden	86
4.10. 9.	Negieren Uebertragsflag	86
4.11.	Unterprogrammaufruf und -rueckkehr	87
4.11. 1.	Unterprogrammaufruf	87
4.11. 2.	Rueckkehr vom Unterprogramm	89
4.11. 3.	Rueckkehr von der Unterbrechungsbehandlung	91
4.12.	CPU-Steuerbefehle	91
4.12. 1.	Koprozessor-Kommando mit Adresse	91
4.12. 2.	Prozessorhalt	92

		Seite
4.12. 3.	Synchronisation mit externer Hardware	93
4.12. 4.	Prozessorinterrupt	93
4.12. 5.	Unterbrechung bei Ueberlauf	94
4.13.	Ein-/Ausgabebefehle	95
4.13. 1.	Eingabe Byte oder Wort	95
4.13. 2.	Ausgabe Byte oder Wort	96
4.14.	Korrekturbefehle	97
4.14. 1.	ASCII-Korrektur nach Addition	97
4.14. 2.	ASCII-Korrektur vor Division	98
4.14. 3.	ASCII-Korrektur nach Multiplikation	99
4.14. 4.	ASCII-Korrektur nach Subtraktion	99
4.14. 5.	Dezimalkorrektur nach Addition	100
4.14. 6.	Dezimalkorrektur nach Subtraktion	101
4.15.	Wandelbefehle	102
4.15. 1.	Umwandlung Byte in Wort	102
4.15. 2.	Umwandlung Wort in Doppelwort	102
4.16.	Zeichenkettenbefehle	103
4.16. 1.	Vergleich Bytekette/Wortkette	103
4.16. 2.	Laden Byte- oder Wortkette in den Akkumulator	104
4.16. 3.	Blocktransport im Speicher	105
4.16. 4.	Suchen Akkumulatorinhalt im Speicher	106
4.16. 5.	Speicher mit Inhalt Akkumulator laden	107
Anhang A	CPU-Befehlsliste	108

1. CPU-Befehlsbeschreibung

1.1. Programmiermodell und Registerarchitektur des K1810 WM86

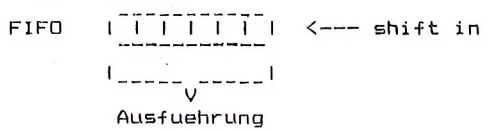
1.1. Allgemeines

Der Prozessorschaltkreis K1810 WM86 ist ein leistungsfaehiger zum I8086 kompatibler 16 Bit Mikroprozessor. Er realisiert byte- (8 Bit) und wortweise (16 Bit) Speicherzugriffe. Die Prozessorbefehle bearbeiten die Einheiten Bit, Byte (8 Bit), Wort (16 Bit), Doppelwort (2 Worte) und Zeichenketten (Bloecke).

Der Speicheradressraum ist durch eine Adressbreite von 20 Bit in der Groesse 1 MByte (1024 KByte) gegeben. Daten und Kode sind im Adressraum dynamisch verschieblich.

Die Anzahl der Bytes liegt je nach Prozessorbefehl zwischen 1 und 6 Byte. Die Adressierung eines Wortes im Speicher erfolgt in der Weise, dass die Adresse das niederwertige Byte des Wortes adressiert und das hoeherwertige Byte des Wortes an der naechsten Adressstelle (Adresswert + 1) steht.

Der Prozessor arbeitet mit einem internen 6 Byte-FIFO-Speicher fuer den zu verarbeitenden Kode. Der Kodestrom wird zeitlich vor der eigentlichen Befehlsausfuehrung wortweise in den FIFO des Prozessors geladen.



Diese Einrichtung erhoeht fuer im Speicher aufeinanderfolgende Befehle den Systemdurchsatz. Dies gilt insbesondere fuer Befehle mit ungerader Bytezahl (wortweises Laden in den FIFO!). Fuer Programmverzweigungsbefehle (Spruenge, Unterprogrammaufrufe) ist fuer Zeitbetrachtungen zu beachten, dass die bereits im FIFO zwischengespeicherten Folgebefehle ungueltig werden und durch neue ersetzt werden muessen.

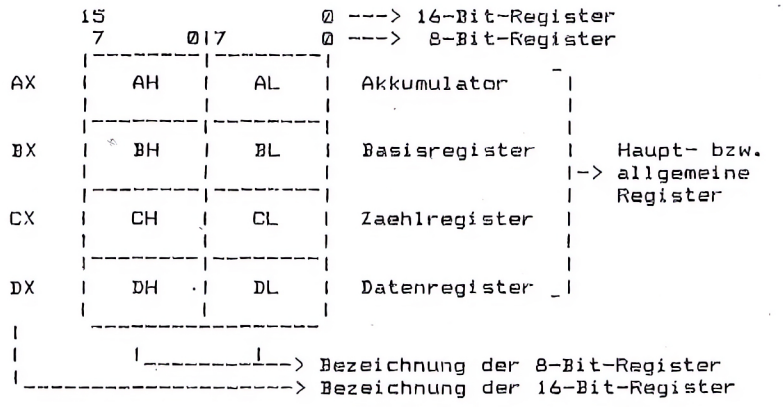
1.2. Registerstruktur

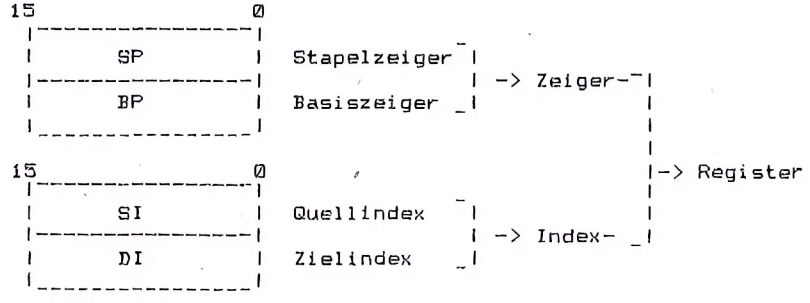
Der K1810 WM86 verfuegt ueber 14 16-Bit-Register.

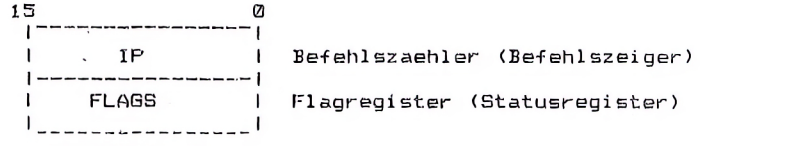
- 4 Hauptregister
- 2 Indexregister
- 1 Befehlszaehler (Befehlszeiger)
- 1 Flagregister (Statusregister)
- 4 Segmentregister
- 1 Stapelzeiger (Stackpointer)
- 1 Basiszeiger

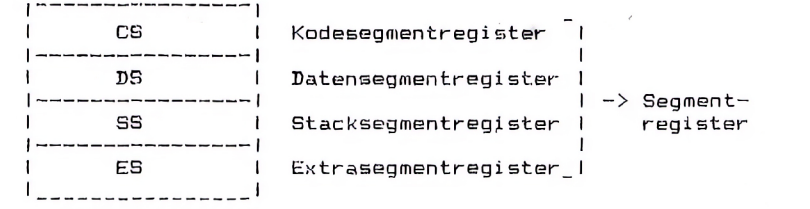
*** CPU - BEFEHLSBESCHREIBUNG ***

Die dem Programmierer zugaenglichen Registerstrukturen sind:









Haeufig benutzte englischsprachige Bezeichnungen fuer die Register sind:

CX	Zaehlregister	count
DX	Datenregister	data
SI	Quellindex	source index
DI	Zielindex	destination index
SP	Stapelzeiger	stack pointer
BP	Basiszeiger	base pointer
IP	Befehlszeiger	instruction pointer

1.3. Speicheradressen, Segmente

Ein Speicherplatz im 1 MByte-Speicheradressraum wird durch eine 20 Bit breite absolute Adresse adressiert. Der Prozessor K1810 WM86 errechnet Speicheradressen durch Kombination einer effektiven Adresse mit einer Segmentbasisadresse aus einem der vier Segmentregister. Die Segmentbasisadresse definiert den Beginn bzw. die Untergrenze eines 64 KByte grossen Segmentes. Die effektive Adresse ist eine relative Adresse innerhalb eines Segmentes, bezogen auf die Segmentbasis. Folglich stellt sie einen Offset (Verschiebung) dar. Fuer die Adressdarstellung innerhalb eines 64 KByte-Segmentes ist eine 16-Bit-Adresse notwendig.

Die Segmentbasisregister, selbst 16 Bit breit, verweisen auf den Anfang von Speichergruppen der Groesse 16 Byte, beginnend ab der absoluten Speicheradresse 0. Die Segmentregister fungieren also beim K1810 WM86 als Basisregister, die auf jeden Speicherplatz zeigen koennen, dessen Adresse durch 16 teilbar ist.

Die absolute Speicheradresse, auch als physikalische Speicheradresse bezeichnet, ergibt sich als Summe aus der Segmentbasisadresse (die Segmentregister liefern die Bits 4 bis 19, die Bits 0 bis 3 sind gleich 0) und dem 16-Bit-Offset als effektive Adresse.

```
Segmentbasisadresse       :   xxxx xxxx xxxx xxxx 0000
effektive Adresse (Offset) : + 0000 yyyy yyyy yyyy yyyy
-------------------------------------------------------
absolute Adresse          :   xxxx zzzz zzzz zzzz yyyy (20 Bit)
```

Eine absolute Adresse besteht, wie der Darstellung zu entnehmen ist, aus zwei Teiladressen:

- dem Inhalt des Segmentbasisregisters, d.h. der Segmentbasisadresse und
- der effektiven Adresse (Offset) innerhalb des Segments.

Die effektive Adresse kann sich aus Registerwerten und anderen

Werten zusammensetzen.

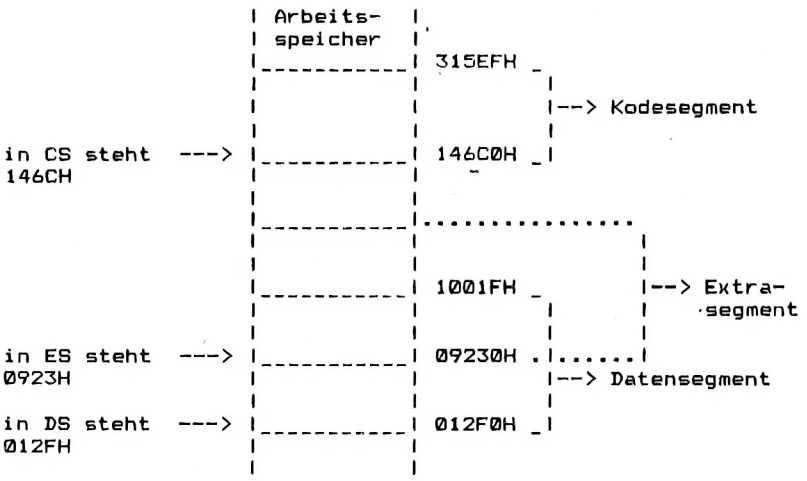
Beispiel:

```
MOV  AL,1000H ; 1000H = Direktadresse
```

adressiert die Speicherstelle mit der effektiven Adresse 1000H im Datensegment;

Wenn DS = 2000H ist, wird auf die Speicherstelle mit der absoluten Adresse 21000H zugegriffen und der Inhalt in das Register AL geladen.

Das nachfolgende Schema stellt die Segmenteinteilung im Arbeitsspeicher dar:



Da der Prozessor aber ueber 4 Segmentregister verfuegt, kann ein Programm zu jedem beliebigen Zeitpunkt gleichzeitig 4 Speichersegmente definieren und benutzen.

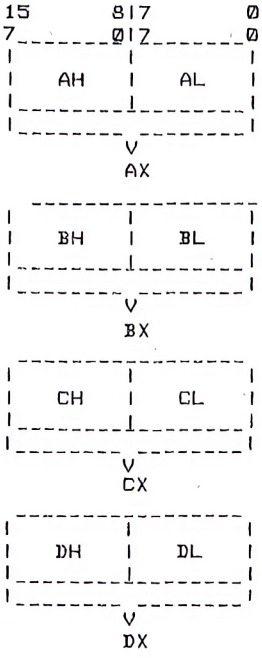
Die Segmente koennen einander ueberlappen oder auch nebeneinander liegen, d.h. die Segmentregister koennen beliebige Werte enthalten. Deshalb ist der Speicher nicht in 64 KByte-Bloecke physikalisch unterteilt. Die Segmentregister muessen auch nicht Speicherbereiche auswaehlen, die sich nicht ueberlappen. Jedes Segmentregister gibt nur die Untergrenze eines Segmentes an, welches sich irgendwo im adressierbaren Speicher befinden kann. Die Segmente koennen nebeneinander liegen, unmittelbar aneinander grenzen bzw. sich auch ueberlappen.

1.4. Die Register des K1010 WM86

1.4.1. Hauptregister/allgemeine Register

Die Hauptregister koennen uneingeschraenkt in arithmetischen, logischen und E/A-Operationen benutzt werden. Daneben besitzt jedes Hauptregister eine oder mehrere Sonderfunktionen (zugewiesene spezielle Bedeutung).

Jedes dieser vier 16-Bit-Register (AX, BX, CX und DX) stellt eine Kombination von zwei 8-Bit-Registern dar.



Durch die Aufteilung dieser 16-Bit-Register in 8-Bit-Register lassen sich 8-Bit-Operationen durchfuehren. Der Vorteil besteht darin, dass Operationen mit 8-Bit-Registern in der Regel weniger Zeit benoetigen. Ausserdem besitzen sie einen kuerzeren Maschinenkode als mit 16-Bit-Registern. Damit wird weniger Speicherplatz benoetigt.

Beispiel:

Abwaertszaehlen eines Zaehlers von 250 dezimal auf 0. Es genuegt ein 8-Bit-Register, um diesen Vorgang auszufuehren.

Sonderfunktionen der Hauptregister:

Register AX: Alle ueber E/A-Befehle zu handhabenden Daten werden ueber dieses Register ein- bzw. ausgegeben. Sind in einigen Befehlen Direktwerte als Operand kodiert, bringt die Durchfuehrung der Operation ueber das Register AX einen kuerzeren Maschinenkode.

Register BX: Fungiert als Basisadressregister, d.h., sein Inhalt wird bei der Bildung von Speicheradressen verwendet.

Register CX: Sonderfunktion; Verwendung als Zaehlregister; Der Inhalt dieses Registers wird bei der wiederholten Ausfuehrung von Zeichenkettenbefehlen und bei Verschiebeoperationen um mehr als eine Bitposition abwaerts gezaehlt.

Register DX: Es enthaelt bei einigen E/A-Operationen die Adresse des jeweiligen E/A-Kanals (Port-Adresse). Diese Funktion kann kein anderes Register uebernehmen.
Eine weitere Sonderaufgabe des Registers DX besteht in der Aufnahme von Operanden und Ergebnissen bei Multiplikationen und Divisionen.

1.4.2. Zeigerregister

Ueber die Zeigerregister SP und BP werden Speicherplaetze im Stacksegment adressiert. Ausserdem koennen in ihnen Operanden fuer 16 Bit breite arithmetische und logische Operationen gespeichert werden.

Das Register SP (Stapelzeiger oder Stackpointer) ermoeglicht im Speicher den Aufbau eines Stacks. Die Adressierung der Speicherplaetze erfolgt durch das Register SP in Verbindung mit dem Register SS.

Das Register BP erlaubt den Zugriff zu Daten im Stacksegment, in der Regel zu Parametern, die ueber den Stack weitergegeben werden. Der Inhalt des Registers BP wird ebenfalls mit dem Inhalt des Registers SS kombiniert.

SP, BP adressieren Speicherplaetze im Stacksegment. Die Register enthalten Offsets zur Adressberechnung innerhalb des Stacksegmentes. BP kann auch eine Basisadresse zur Adressberechnung innerhalb des Stacksegmentes enthalten.

1.4.3. Indexregister

Ueber die Indexregister SI und DI werden meist Speicherplaetze im Datensegment adressiert, die Zeichenketten enthalten. Werden Zeichenkettenbefehle angewendet, adressiert DI immer Speicher-

plaetze im Extrasegment. Ausserdem koennen in den Indexregistern Operanden fuer 16 Bit breite arithmetische und logische Operationen gespeichert werden.

SI, DI adressieren immer Speicherplaetze im Datensegment. Die Register enthalten Offsets oder Basisadressen zur Adressberechnung innerhalb des Segmentes.

Ausnahme: Bei Zeichenkettenbefehlen adressiert DI Speicherplaetze im Extrasegment.

1.4.4. Befehlszeiger

Der Befehlszeiger IP enthaelt immer die Adresse desjenigen Befehls, der als naechster ausgefuehrt wird. IP stellt immer den Offset bzgl. des CS-Registerinhaltes (Kodesegmentbasisadresse) dar.

1.4.5. Segmentregister

Bei der Berechnung einer Speicheradresse wird, ausser bei der Adressierung von Interruptvektoren, der Inhalt eines der 4 Segmentregister einbezogen. Jedes Segmentregister definiert im Speicher den Beginn eines 64 KByte grossen Bereiches, ein Segment. Aufgrund der 4 Segmentregister kann ein Programm gleichzeitig mit maximal 4 Segmenten arbeiten. Es wird zwischen Kode-, Daten-, Stack- und Extrasegment unterschieden.

Register CS: Vor dem Einlesen eines Befehls wird der Inhalt des Kodesegmentregisters CS mit dem Inhalt des Befehlszeiger IP kombiniert, um die Adresse des Speicherplatzes, an der der Befehl beginnt, zu ermitteln. Der Befehlszeiger IP adressiert damit relativ zum Inhalt des Kodesegmentregisters CS.

Register DS: In der Regel werden alle Operanden relativ zum Inhalt des Datensegmentregisters DS adressiert.
Es existieren jedoch folgende Ausnahmen:

1. Wird das Register BP zur Adressberechnung genutzt, ist automatisch das Stacksegment adressiert. Die Segmentbasisadresse ist dann der Inhalt des Registers SS.
2. Zeichenkettenbefehle, die zur Bildung von Operandenadressen den Inhalt des Registers DI benutzen, verwenden als Segmentbasisadresse den Inhalt des Registers ES.

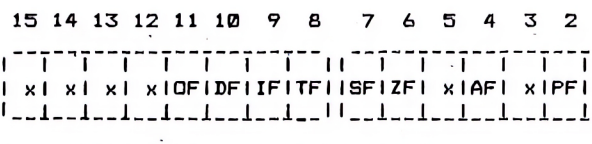
Register SS: In allen Operationen, die Adressen unter Verwendung des Inhaltes der Register SP oder BP bilden, wird bei der Bildung der absoluten Adresse der Inhalt des Stacksegmentregisters verwendet. Alle stackorientierten Befehle wie PUSH, POP, CALL, RET und INT werden nur in Verbindung mit dem Register SS ausgefuehrt.

Register ES: Bestimmte Zeichenkettenbefehle verwenden bei der Berechnung der Operandenadresse den Inhalt des Registers DI. Die so berechneten Adressen werden relativ zum Inhalt des Extrasegmentregisters verwendet.

1.4.6. Flagregister

Der K1810 WM86 verfuegt ueber ein 16-Bit-Flagregister, auch Statusregister genannt. Sein Inhalt wird auch als Prozessorstatuswort bezeichnet.

Die einzelnen Bits besitzen folgende Bedeutung:



x - reserviert, normalerweise 0

OF - Ueberlauf bei Operationen mit Vorzeichen
(6.-->7. Bit bei 8-Bit Operanden und 14.-->15. Bit bei 16-Bit Operanden)

DF - Richtungsflag
(entscheidet bei Zeichenkettenbefehlen die Abarbeitungsrichtung aufwaerts bzw. abwaerts)

IF - Interruptflag
(INT fuer externe Interrupts)

TF - Einzelschrittflag
(setzen TF erzeugt in der CPU Einzelschrittabarbeitung)

SF - Vorzeichenflag

ZF - Nullflag

AF - Hilfsuebertragsflag
(fuer BCD-Arithmetik bei Arbeit mit Tetraden)

PF - Paritaetsflag

CF - Uebertragsflag

Die Bits 0 bis 7 koennen mittels CPU-Befehlen nach dem Register AH transportiert, bzw. von dort geladen werden.

Haeufig benutzte englischsprachige Bezeichnungen fuer die Flags sind:

OF	Overflow - Flag
DF	Direction - Flag
IF	Interrupt - Flag
TF	Trap - Flag
SF	Sign - Flag
ZF	Zero - Flag
AF	Auxiliary carry - Flag
PF	Parity - Flag
CF	Carry - Flag

2. Metasprache

2.1. Allgemeine Elemente

[reg]	Inhalt Register
[flags]	Inhalt Flagregister
[mem]	Inhalt des adressierten Speicherplatzes
mem	Adresse des Speicherplatzes (Offset)
{...} {...|...}	Auswahl eines innerhalb der Klammer stehenden Ausdruckes; beide Formen sind gleichbedeutend.
[...]	wahlfrei; der Ausdruck kann geschrieben werden

Kleinbuchstaben
: Ein kleingeschriebener Text ist ein Syntaxterminus bzw. Syntaxausdruck. Er muss durch eine konkrete Zeichenkette ersetzt werden.

Grossbuchstaben
: Bestandteil des Befehls, der unveraendert uebernommen werden muss.

&	UND - Verknuepfung (Konjunktion)
|	ODER - Verknuepfung (Disjunktion)
||	Exklusiv - ODER - Verknuepfung (Antivalenz)
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
mod	Modulo - Rechnung

=	gleich
<	kleiner
≤	kleiner gleich
>	groesser
≥	groesser gleich
:=	ist definiert zu (ergibt sich aus ...)

2.2. Operandenelemente

reg8	Byteregister: AL, BL, CL, DL, AH, BH, CH, DH
reg16	Wortregister: AX, BX, CX, DX, SP, BP, SI, DI
sreg	Segmentregister: CS, SS, DS, ES
imm8	Direktwert (Byte)
imm16	Direktwert (Wort)
mem8	Konstantenausdruck (Bytewert) oder E/A- (Port-) Adresse
mem16	Direktadresse (numerisch angegebene Adresse) oder Konstantenausdruck (Wort) mem16 kann als Direktadresse in den Formen: - faradr: Adresswert ausserhalb des aktuellen Segments - nearadr: Adresswert innerhalb des aktuellen Segments auftreten.
memi8, memi16	Indexspezifikation (Index- und bzw. oder Basisregisteradressierung unter Verwendung von BX, BP, DI, SI)
memb8, memb16	nur Basisregisteradressierung (BX oder BP) {memb8 | memb16} = label {[BX] | [BP]}
flags	Flagregister
sdisp	Zieladresse mit Kurzdistanz im Bereich von - 128 ≤ x ≤ + 127 Byte
op1	linker (1.) Operand
op2	rechter (2.) Operand
cond	Bedingungsangabe

ext8 Erweiterung des Akkumulators bei Multiplikation und Division (bei Byteoperationen = Register AH)

ext16 Erweiterung des Akkumulators bei Multiplikation und Division (bei Wortoperationen = Register DX)

2.3. Flageinstellung

X : Flag wird im Ergebnis der Operation beeinflusst

. : Flag wird nicht beeinflusst

1 : Flag wird gesetzt

0 : Flag wird geloescht

U : Flagzustand undefiniert

3. Adressierung

3.1. Adressmodi

Der Prozessor K1810 WM86 verwendet 8 Adressierungsverfahren:

Adressierungsart	Format der Adresse	verwendetes sreg
Register (reg16)	Register enthaelt Offsetadresse im Segment, da 16 Bit gross	--
Direktwert (imm16)	direkt angegebener 16-Bitwert	--
Direktadresse (mem16)	16-Bit Speicheradresse (Offset)	DS
Register-indirekt-Modus	[BX] [BP] [DI] [SI]	DS SS DS DS
Basisadressmodus	label [BX] label [BP]	DS SS
Indexadressmodus	label [SI] label [DI]	DS DS
Basis- und Index-adressmodus kombiniert	[label] [BX+SI] | [label] [BX][SI] [label] [BX+DI] | [label] [BX][DI] [label] [BP+SI] | [label] [BP][SI] [label] [BP+DI] | [label] [BP][DI]	DS DS SS SS

3.1.1. Registeradressierung

Die einfachste Adressierungsart ist in Befehlen mit einem Operanden gegeben, der in einem Register steht. Hierbei handelt es sich nicht um eine Adressierung im eigentlichen Sinne. Die Registeradressierung wird hier als Adressierungsart eingestuft, um das Verstaendnis der Wirkung der 2-Operandenbefehle zu erleichtern. Bei ihnen muss (mit 2 Ausnahmen) ein Operand in einem Register stehen.

Beispiel:

```
INC  AX
```

3.1.2. Adressierung ueber Direktwert

Bei dieser Adressierungsart ist der 1. Operand in einem Register abgespeichert. Der 2. Operand besteht dann aus dem Direktwert.

Beispiel:

```
ADD  AX,1024H
```

Dieser Befehl veranlasst den Prozessor, einen Additionsbefehl auszufuehren, der 1024H zum Inhalt des Registers AX addiert. Dabei ist zu beachten, dass die 8 niederwertigen Bits des 16-Bit-Direktwertes in dem Speicherplatz mit der niedrigeren Adresse abgespeichert und die 8 hoeherwertigen Bits im naechsten Byte des Befehlskodes folgen.

3.1.3. Adressierung ueber Direktadresse

Direkte Adressierung liegt dann vor, wenn ein Offset zum Inhalt des Datensegmentregisters DS (Normalfall) addiert wird. Die Summe aus Offset und Inhalt des Datensegmentregisters stellt die absolute Adresse im Speicher dar.

Die Operandenadresse ist dabei wie unter Pkt. 3.1.2. abgespeichert.

Beispiel:

```
MOV  AX,WERT
```

Es wird das Wort, dass sich auf dem Speicherplatz befindet, der sich aus dem Inhalt von DS und dem Offset WERT ergibt, in das Register AX geladen.

3.1.4. Register-indirekt-Modus

Bei dieser Adressierungsart werden unterschieden:

a) Indirekte Adressierung ueber Indexregister SI oder DI
b) Adressierung relativ zum Register BX
c) Adressierung relativ zum Register BP

a) Indirekte Adressierung ueber Indexregister

Eine der beiden Indexregister SI oder DI enthaelt den Offset des Operanden. Es wird somit indirekt ueber die Indexregister auf den Speicherplatz zugegriffen.

b) Adressierung relativ zum Register BX

Im Register BX steht der Offset des Operanden. Der Inhalt des Registers BX wird auch oft als Basisadresse in weiteren Adressierungsarten verwendet.

c) Adressierung relativ zum Register BP

Unter bestimmten Bedingungen ist es guenstig, wenn Daten im Stacksegment erreichbar sind, ohne dass der Inhalt des Stackpointers veraendert werden muss. Dazu wird dann die Adressierung relativ zum Register BP verwendet.

3.1.5. Basisadressmodus

Diese Adressierungsart verwendet das Register BX als Basisadresse fuer Daten im Datensegment und das Register BP fuer Daten im Stacksegment.

Die effektive Adresse wird dann wie folgt gebildet:

- bei Verwendung BX: label + BX
- bei Verwendung BP: label + BP

Die absolute Adresse wird dann durch die Addition der Segmentbasisadresse (DS bei BX und SS bei BP) ermittelt (siehe Pkt. 3. Schema Bildung absolute Adresse !).

3.1.6. Indexadressmodus

Die Register SI oder DI werden zur Bildung der effektiven Adresse verwendet.

Sie wird auf folgendem Wege ermittelt:

- bei Verwendung SI: label + SI
- bei Verwendung DI: label + DI

3.1.7. Basis- und Indexadressmodus kombiniert

Bei dieser Adressierungsart werden beide Adressierungsarten, die unter Pkt. 3.1.5. und 3.1.6. beschrieben wurden, kombiniert angewendet. Dabei ist zu beachten, dass auch hier bei Verwendung des Registers BP sich auf das Stacksegment bei der Bildung der absoluten Adresse bezogen wird, sonst auf das Datensegment.

Die effektive Adresse wird wie folgt ermittelt:

- bei Verwendung von BX und SI: [label +] BX + SI
- bei Verwendung von BX und DI: [label +] BX + DI
- bei Verwendung von BP und SI: [label +] BP + SI
- bei Verwendung von BP und DI: [label +] BP + DI

3.2. Befehlsaufbau

Es werden 3 Befehlstypen unterschieden:

- ohne Operanden
- mit einem Operanden
- mit zwei Operanden, wobei der 1. Operand in der Regel der Zieloperand und der 2. Operand der Quelloperand ist.

Weiterhin ist zu beachten, dass keine direkten Transporte von Speicher zu Speicher moeglich sind. Es muessen immer Register als Zwischenspeicher verwendet werden.

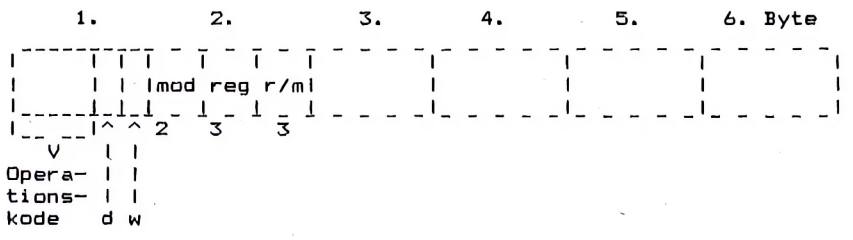
Allgemeiner Aufbau einer Assemblerbefehlszeile:

[name] <mnemonik> [<operanden>] [;<kommentar>]

Befehlsstruktur:

Der Kode des Befehls kann aus 1 bis 6 Byte bestehen.

Allgemeiner Aufbau:



Erklaerung:

d: Richtung (direction): d = 1 in REG ist Zielregister kodiert
d = 0 in REG ist Quellregister kodiert

w: Wort/Byte (word)
w = 1 --> Wort
w = 0 --> Byte

mod: Modifikation (modification)

reg: Register (register)
definiert ein Register oder ist eine Erweiterung des Operationskodes

r/m: Register/Speicherfeld (register/memory)

Kodierung Modifikationsfeld (mod):

mod 00 Speichermodus, keine Verschiebung
01 Speichermodus, 8-Bit-Verschiebung
10 Speichermodus, 16-Bit-Verschiebung
11 Registermodus (nur Arbeit zwischen Registern), ausser wenn r/m = 110 ist, dann 16-Bit-Verschiebung

Kodierung reg:

Dieses Feld definiert in Verbindung mit dem w-Bit ein 8- bzw. 16-Bit-Register, welches einem Befehlsoperanden entspricht.

reg	w = 0	w = 1	Segment-register
000	AL	AX	ES
001	CL	CX	CS
010	DL	DX	SS
011	BL	BX	DS
100	AH	SP	
101	CH	BP	
110	DH	SI	
111	BH	DI	

Dieses Feld wird bei bestimmten Befehlen auch zur Erweiterung des Operationskodes und zur Bestimmung des Operationstyps benutzt.

Register/Speicher - Feld (r/m):

Dieses Feld ist in Verbindung mit dem Modifikationsfeld mod zu betrachten. Im Registermodus bestimmt r/m den 2. Registeroperanden; im Speichermode jedoch die effektive Adresse des Speicheroperanden.

Registermodus		Effektive Adressenbestimmung			
mod = 11		mod = 00	mod = 01		mod = 10
r/m	w=0	w=1			
000	AL	AX	[BX][SI]	disp8 [BX][SI]	disp16 [BX][SI]
001	CL	CX	[BX][DI]	disp8 [BX][DI]	disp16 [BX][DI]
010	DL	DX	[BP][SI]	disp8 [BP][SI]	disp16 [BP][SI]
011	BL	BX	[BP][DI]	disp8 [BP][DI]	disp16 [BP][DI]
100	AH	SP	[SI]	disp8 [SI]	disp16 [SI]
101	CH	BP	[DI]	disp8 [DI]	disp16 [DI]
110	DH	SI	--	disp8 [BP]	disp16 [BP]
111	BH	DI	[BX]	disp8 [BX]	disp16 [BX]

disp8 : 8-Bit-Verschiebung (Displacement)
disp16: 16-Bit-Verschiebung

Die Bytes 3 bis 6 des Befehls enthalten den Verschiebungswert des Speicheroperanden und/oder des aktuellen Wertes eines unmittelbaren konstanten Operanden oder eine effektive Adresse.

4. Beschreibung der Befehle

4.1. Praefixe

Praefixe werden immer vor den eigentlichen Befehl oder den entsprechenden Operanden geschrieben.

4.1.1.1. Wiederholungspraefixe

Die Wiederholungspraefixe koennen fuer die Zeichenkettenbefehle Verwendung finden. Sie ueberfuehren einfache Kettenoperationen in wiederholbare, solange zugeordnete Bedingungen nicht erfuellt sind.

Es werden folgende Wiederholungspraefixe unterschieden:

REP

Anwendung fuer die Zeichenkettenbefehle LODS, MOVS, STOS.

Die spezifizierte Kettenoperation wird so lange wiederholt, bis das Zaehlregister [CX] = 0 wird.

CX wird nach jedem Durchlauf um 1 dekrementiert.

Beispiel:

```
        MOV  SI,OFFSET QUELL_KETTE
        MOV  DI,OFFSET ZIEL_KETTE
        MOV  CX,LAENGE_QUELLE
REP     MOVSB
```

REPE, REPNE, REPZ, REPNZ

Anwendung fuer die Zeichenkettenbefehle CMPS und SCAS.
Die Anzahl der Wiederholungen ist hier abhaengig vom Z-Flag und [CX].

Der Befehl wird solange wiederholt, wie die Bedingung (E, NE, Z, NZ) erfuellt oder [CX] nicht 0 geworden ist.

Beispiel:

```
        MOV  SI,OFFSET KETT1
        MOV  DI,OFFSET KETT2
        MOV  CX,LAENGE_KETT1
REPE    CMPSB
```

4.1.1.2. Bus-Lock-Signal fuer einen nachfolgenden Befehl aktivieren

Schreibweise:

LOCK

LOCK wird dem Befehl vorangestellt. Der LOCK-Praefix fuehrt keine Operationen aus und beeinflusst keine Flags. Er kann mit Segment-Overrideoperator und / oder REP - Praefix kombiniert werden.

Ein spezieller Ein-Byte-LOCK-Praefix kann jedem Befehl vorangestellt werden. Es veranlasst den Prozessor das Bus-Lock-Signal fuer den folgenden Befehl zu beanspruchen.

Mit Hilfe des LOCK-Praefixes wird das Hardware-Pinnlock fuer die gesamte Ausfuehrungszeit des einen nachgestellten Befehls aktiviert.

LOCK wird in Mehrprozessorsystemen benutzt, um kritische Bereiche waehrend einem sogenannten Read-Modify-Zyklus gegen Zugriffe zu sichern.

4.2. Ladebefehle

4.2.1. Uebertragen von Daten

Befehl:	MOV	Move
Schreibweise:	MOV op1,op2	

Flags: keine Beeinflussung

Operandenkombinationen:

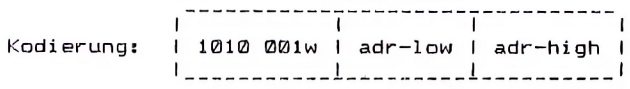
Typ	op1	op2
1	memb16 memb8	AX AL
2	AX AL	memb16 memb8
3	sreg sreg	reg16 (ausser CS) (mem16|memi16) " "
4	reg16 (mem16|memi16)	sreg sreg
5	reg8 reg16 reg8 (mem8|memi8 reg16 (mem16|memi16)	reg8 reg16 (mem8|memi8) reg8 (mem16|memi16) reg16
6	reg8 reg16	imm8 imm16
7	(mem8|memi8) (mem16|memi16)	imm8 imm16

Befehlswirkung: [op1] := [op2]

Der Inhalt des rechten Operanden [op2] wird in den linken Operanden [op1] uebertragen. Der rechte Operand bleibt erhalten.

Es werden 7 Typen des Befehls **MOV** unterschieden:

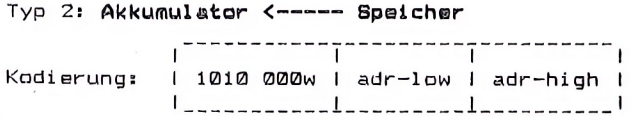
Typ 1: **Speicher <----- Akkumulator**



Beispiele:

```
MOV   ALPHA_MEM,AX
MOV   BYTE PTR GAMMA,AL
MOV   CS:BYTE PTR DATUM,AL
MOV   ES:FELD[BX][SI],AX
```

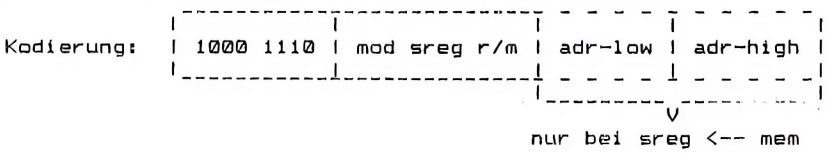
Typ 2: **Akkumulator <----- Speicher**



Beispiele:

```
MOV   AX,WORD PTR ALPHA_MEM
MOV   AL,BYTE PTR GAMMA
MOV   AX,ES:FELD[BX][SI]
MOV   AL,SS:SPEICH_BYTE
```

Typ 3: **Segmentregister <--- Speicher oder Register**



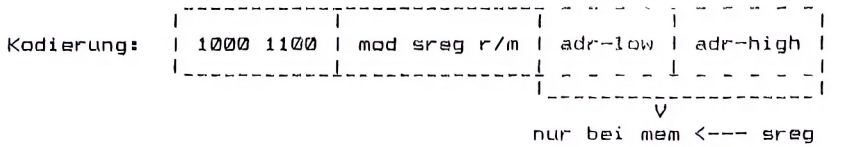
Beispiele:

```
MOV   ES,DX
MOV   DS,AX
MOV   SS,BX
MOV   ES,SS:NEW_WORD[DI]
```

Beachte:

Das Kodesegmentregister CS darf als Zielregister nicht verwendet werden.

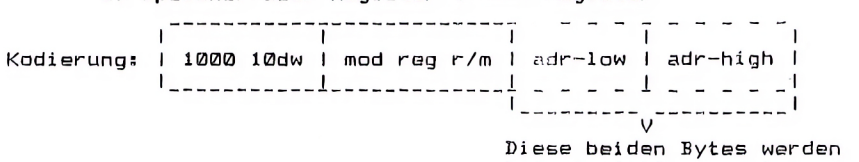
Typ 4: **Speicher oder Register <--- Segmentregister**



Beispiele:

```
MOV   DX,ES
MOV   SS:NEW_WORD[DI],ES
MOV   CS,AX              ; CS hier erlaubt
```

Typ 5: **a) Register <----- Register**
b) Register <----- Speicher oder Register
c) Speicher oder Register <----- Register



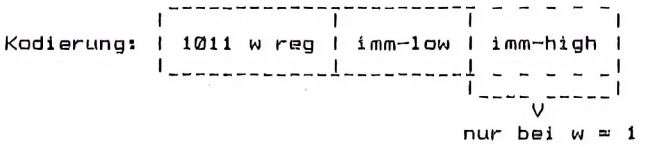
Beispiele:

```
a) MOV   AX,BX
   MOV   CL,DH
   MOV   CX,DI

b) MOV   AX,MEM_VALUE
   MOV   DX,WORD PTR FELD[SI]
   MOV   DI,MEM[BX][SI]

c) MOV   WORD PTR FELD[DI],DX
   MOV   MEM_VALUE,AX
   MOV   [BX][SI],DI
```

Typ 6: **Register <----- Direktwert**

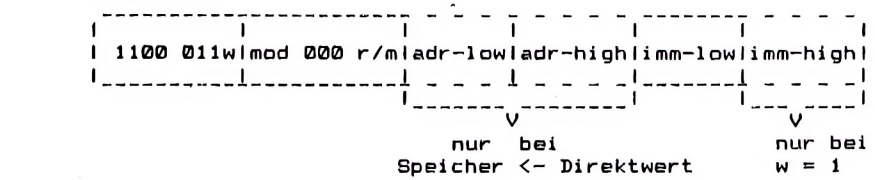


Beispiele:

```
MOV   AX,77
MOV   DI,618
MOV   BL,2
```

Typ 7: **Speicher oder Register <----- Direktwert**

Kodierung:



Beispiele:

```
MOV   DS:MEM_WORD[BP],3897
MOV   BYTE PTR [DI],66
MOV   BX,10655
```

4.2.2. Laden Datensegmentregister

Befehl:	LDS	Load data segment register
Schreibweise:	LDS op1,op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2
1	reg16	mem16
	reg16	memi16

Befehlswirkung: [op1] := [op2]
[DS] := [op2+2]

Diese Operation ermoeglicht es, das Datensegmentregister und ein weiteres Wortregister mittels eines Befehls zu laden. Der zweite Operand muss ein Doppelwortspeicheroperand sein [op2].

Befehlsablauf:

1. Das niederwertig adressierte Wort des Doppelwortspeicheroperanden wird in das spezifizierte 16-Bitregister geladen.

   [op1] := [op2]

2. Das hoeherwertig adressierte Wort des Doppelwortspeicheroperanden wird in das Datensegmentregister (DS) geladen.

   [DS] := [op2+2]

Kodierung:

1100 0101	mod reg r/m	adr-low	adr-high

Beispiele:

```
LDS   BX,ADDR_TABLE[SI]
LDS   SI,NEWSEG[BX]
LDS   AX,MEM_DWORD
```

4.2.3. Laden Extrasegmentregister

Befehl:	LES	Load extra segment
Schreibweise:	LES op1,op2	

Flags: keine Beinflussung

Operandenkombinationen:

Typ	op1	op2
1	reg16	mem16
	reg16	mem16

Befehlswirkung: [op1] := [op2]
[ES] := [op2+2]

Diese Operation ermoeglicht es, das Extrasegmentregister und ein weiteres Wortregister mittels eines Befehls zu laden. Der zweite Operand muss ein Doppelwortspeicheroperand sein [op2].

Befehlsablauf:

1. Das niederwertig adressierte Wort des Doppelwortspeicheroperanden wird in das spezifizierte 16-Bitregister geladen.

   [op1] := [op2]

2. Das hoeherwertig adressierte Wort wird in das Extrasegmentregister (ES) geladen.

   [ES] := [op2+2]

Kodierung:	1100 0100	mod reg r/m	adr-low	adr-high

Beispiele:

```
LES   AX,ADDR_TAB[BX][SI]
LES   SI,DOPPEL_WORT[BX]
```

4.2.4. Laden effektive Adresse

Befehl:	LEA	Load effective address
Schreibweise:	LEA op1,op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2
1	reg16	mem16
	reg16	memi16

Befehlswirkung: [op1] := op2

Der Befehl transportiert die Offsetadresse des Quelloperanden [op2] in den Zieloperand [op1].

Kodierung:	1000 1101	mod reg r/m	adr-low	adr-high

Beispiele:

```
LEA  BX,VARIABLE
LEA  DX,BETA[BX][SI]
LEA  AX,[BP][DI]
```

4.3. Austauschbefehl

Befehl	XCHG	Exchange
Schreibweise:	XCHG op1,op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2
1	AX	reg16
2	reg16	reg16
	reg8	reg8
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16

Befehlswirkung: [op1] <-----> [op2]

Der Befehl tauscht den Byte- oder Wortoperanden [op2] mit dem Zieloperanden [op1] aus.

Befehlsablauf:

1. Der Inhalt des linken Operanden wird in einem internen Arbeitsspeicher abgespeichert.

   [areg] := [op1]

2. Der Inhalt des linken Operanden wird durch den Inhalt des rechten Operanden ersetzt.

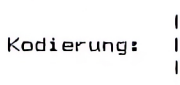
   [op1] := [op2]

3. Der Inhalt des internen Arbeitsspeichers wird in den rechten Operanden eingetragen. Damit ist der Austausch beendet.

   [op2] := [areg]

Es werden 2 Typen des Befehls **XCHG** unterschieden:

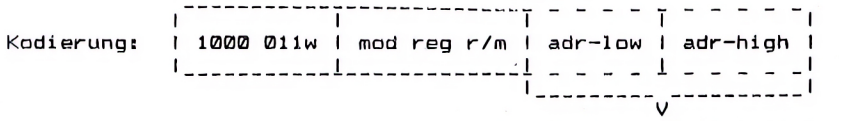
Typ 1: **16-Bit-Akkumulator (AX) <----> 16-Bit-Register**



Beispiele:

```
XCHG    AX,BX
XCHG    AX,CX
```

Typ 2: **Speicher oder Register <----> Register**



Beispiele:

```
XCHG    BL,BL
XCHG    DX,BX
XCHG    BETA_WORD,CX
XCHG    DH,BYTE PTR ALPHA
XCHG    CX,WORD PTR [BX][DI]
```

4.4. Stackbefehle

4.4.1. Stack leeren

Befehl:	POP	Pop word off stack
Schreibweise:	POP op1	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1
1	reg16
2	sreg ausser CS !
3	mem16
	memi16

Befehlswirkung: [op1] := [SP]
SP := SP + 2

POP transportiert einen Wortoperanden aus dem durch das Register SP adressierten Stackelement in den Zieloperanden . Anschliessend wird SP um 2 erhoeht.

Es werden 3 Typen des Befehls **POP** unterschieden:

Typ 1: **Ziel: Wortregister**

Kodierung: | 0101 1 reg |

Beispiele:

```
POP  CX
POP  DX
POP  AX
POP  BX
```

Typ 2: **Ziel: Segmentregister**

Kodierung: | 000 sreg 111 |

Beispiele:

```
POP  DS
POP  ES
```

Achtung ! POP CS ist nicht erlaubt!

Typ 3: Ziel: Speicherplatz

Kodierung:	1000 1111	mod 000 r/m	adr-low	adr-high

Beispiele:

```
POP  ALPHA
POP  BETA[BX]
POP  GAMMA[DI]
POP  EMMA[BX][SI]
```

4.4.2. Flags aus dem Stack laden

Befehl:	**POPF**	**Pop flags off stack**
Schreibweise:	**POPF**	

Flags: OF,DF,IF,TF,SF,ZF,AF,PF,CF: X

Operandenkombinationen: keine Operanden

Befehlswirkung: [flags] := [SP]
SP := SP + 2

POPF transportiert die spezifischen Bits aus dem durch das Register SP adressierten Stackelement in die Flags und erhoeht SP um 2.

Das Fuellen der Flags erfolgt in der Reihenfolge:

Ueberlaufflag	OF <--- Bit 11
Richtungssflag	DF <--- Bit 10
Interruptflag	IF <--- Bit 9
Trap-Flag	TF <--- Bit 8
Vorzeichenflag	SF <--- Bit 7
Nullflag	ZF <--- Bit 6
Hilfsuebertragsflag	AF <--- Bit 4
Paritaetsflag	PF <--- Bit 2
Uebertragsflag	CF <--- Bit 0

Kodierung:	1001 1101

Beispiel:

```
POPF
```

4.4.3. Stack fuellen

Befehl:	PUSH	Push word onto stack
Schreibweise:	PUSH op1	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1
1	reg16
2	sreg CS ist zulaessig!
3	mem16 memi16

Befehlswirkung: SP := SP - 2
[SP] := [op1]

PUSH dekrementiert den Stackpointer um 2 und transportiert ein Wort vom Operanden [op1] zum durch das Register SP adressierten Stackelement, d.h., das Wort wird auf die Spitze des Stacks geschrieben.

Es werden 3 Typen des **PUSH**-Befehls unterschieden:

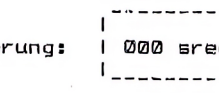
Typ 1: **Quelloperand = Wortregister**

Kodierung: 0101 0 reg

Beispiele:

```
PUSH  AX
PUSH  SI
PUSH  DX
```

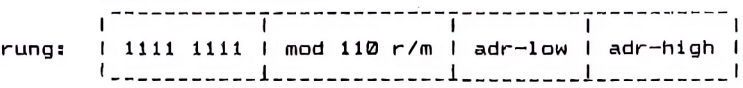
Typ 2: **Quelloperand = Segmentregister**



Beispiele:

```
PUSH  SS
PUSH  ES
PUSH  CS
```

Typ 3: **Quelloperand = Speicher**



Beispiele:

```
PUSH  BETA
PUSH  EMIL[BX]
PUSH  BERTA[BX][DI]
```

4.4.4. Flags in den Stack laden

Befehl:	PUSHF	Push flags onto stack
Schreibweise:	PUSHF	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung: SP := SP - 2
[SP] := [flags]

PUSHF dekrementiert den Stackpointer um 2 und transportiert alle Flags in die entsprechenden Bits des Wortoperanden im durch das Register SP adressierten Stackelement.

Das Fuellen der Bits erfolgt in der Reihenfolge:

Ueberlaufflag	OF --->	Bit 11
Richtungssflag	DF --->	Bit 10
Interruptflag	IF --->	Bit 9
Trap-Flag	TF --->	Bit 8
Vorzeichenflag	SF --->	Bit 7
Nullflag	ZF --->	Bit 6
Hilfsuebertragsflag	AF --->	Bit 4
Paritaetsflag	PF --->	Bit 2
Uebertragsflag	CF --->	Bit 0

Kodierung: | 1001 1100 |

Beispiel:

```
PUSHF
```

4.5. Schleifenbefehle

4.5.1. Schleifenbefehl

Befehl:	**LOOP**	**Loop, or iterate instruction sequence until count complete**
Schreibweise:	**LOOP op1**	

Flags: keine Beeinflussung

Operandenkombinationen: sdisp

Befehlswirkung: [CX] := [CX] - 1
IF [CX] ≠ 0 THEN [IP] := [IP] + sdisp

LOOP dekrementiert das Zaehlregister (CX) um 1 und springt zur Zieladresse (sdisp), falls [CX] nicht Null ist.
Die Zielmarke muss sich, gemessen von diesem Befehl aus, im Bereich von $-128 \leq x \leq +127$ Bytes befinden.
Sobald [CX] den Wert Null erreicht, wird die Schleife verlassen.

Kodierung:	1110 0010	sdisp

Beispiel:

```
            MOV   CX,LENGTH ARRAY
            MOV   AX,0
            MOV   SI,AX

    NEXT:   ADD   AX,ARRAY[SI]
            ADD   SI,TYPE ARRAY
            LOOP  NEXT
              .
              .
              .
```

4.5.2. Bedingte Schleifenbefehle mit Sprung bei Gleich oder Null

Befehl:	bei Gleich: LOOPE bei Null: LOOPZ	Loop on equal Loop on zero
Schreibweise:	LOOPE op1 LOOPZ op1	

Flags: keine Beeinflussung

Operandenkombinationen: sdi ρ

Befehlswirkung:
```
[CX] := [CX] - 1
IF  ([ZF] = 1 & [CX] ≠ 0)
    THEN [IP] := [IP] + sdisp
```

Die Zielmarke muss im Bereich von $-128 \leq x \leq +127$ liegen. **LOOPE** und **LOOPZ** dekrementieren das Zaehlregister (CX). Wenn das Z-Flag (ZF) und das Register CX nicht Null sind, wird zur im Befehl angegebenen Marke verzweigt.

Kodierung:

1110 0001	sdisp

Beispiel:

```
            MOV   CX,LENGTH ARRAY
            MOV   SI,-1

NEXT:       INC   SI
            CMP   ARRAY[SI],0
            LOOPE NEXT
              .
              .
              .
```

4.5.3. Bedingte Schleifenbefehle mit Sprung bei Ungleichheit oder nicht Null

Befehl:	bei Ungleich: LOOPNE bei nicht Null: LOOPNZ	Loop on not equal Loop on not zero
Schreibweise:	LOOPNE op1 LOOPNZ op1	

Flags: keine Beeinflussung

Operandenkombinationen: sdisp

Befehlswirkung:
```
[CX] := [CX] - 1
IF ([ZF] = 0 & [CX] ≠ 0)
   THEN [IP] := [IP] + sdisp
```

Die Zielmarke muss im Bereich von $-128 \leq x \leq +127$ liegen. **LOOPNE** und **LOOPNZ** dekrementieren das Zaehlregister (CX). Wenn das Z-Flag nicht gesetzt und das Register CX nicht Null ist, wird zur im Befehl angegebenen Marke verzweigt.

Kodierung:

1110 0000	sdisp

Beispiel:

```
            MOV     AX,0
            MOV     SI,-1
            MOV     CX,LAENGE
NONZER:     INC     SI
            MOV     AL,BYTE PTR ARRAY1[SI]
            ADD     AL,BYTE PTR ARRAY2[SI]
            MOV     SUM[SI],AX
            LOOPNZ  NONZER
              .
              .
              .
```

4.6. Arithmetikbefehle

4.6.1. Addition ohne Uebertrag

Befehl:	ADD	Addition
Schreibweise:	ADD op1,op2	

Flags: OF, SF, ZF, AF, PF, CF: X

Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

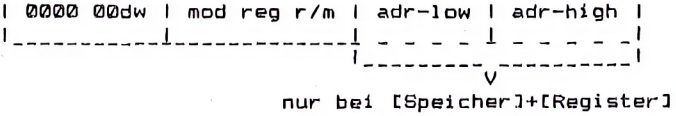
Befehlswirkung: [op1] := [op1] + [op2]

Der Inhalt der beiden Operanden wird addiert und die Summe im linken Operanden abgelegt. Der rechte Operand bleibt erhalten.

Es werden 3 Typen des Befehls **ADD** unterschieden:

Typ 1: **[Speicher oder Register] + [Register]**

Kodierung:

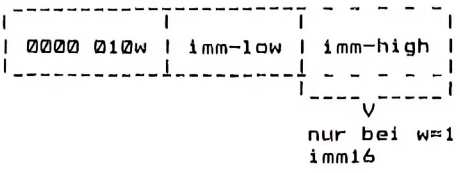


Beispiele:

```
ADD   AL,DH
ADD   BX,DX
ADD   BX,WORD PTR SPEI_WORT
ADD   WORD PTR BETA[DI],AX
ADD   BYTE PTR GAMMA[BP][DI],BL
```

Typ 2: **[Akku] + Direktwert**

Kodierung:

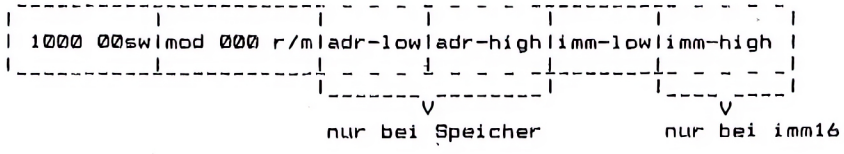


Beispiele:

```
ADD   AL,3
ADD   AX,456
```

Typ 3: **[Speicher oder Register] + Direktwert**

Kodierung:



Beispiele:

```
ADD   BL,90
ADD   CX,9627
ADD   BYTE PTR DELTA[BX][SI],66
ADD   WORD PTR GAMMA[DI],999
ADD   SUMME,6666
```

4.6.2. Addition mit Uebertrag

Befehl:	ADC	Addition with carry
Schreibweise:	ADC op1,op2	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

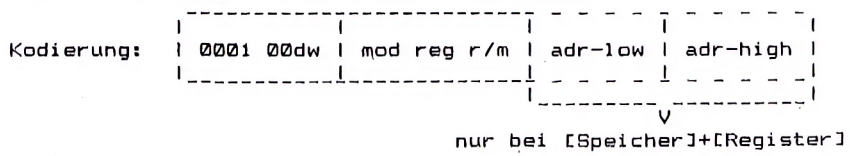
Befehlswirkung: [op1] := [op1] + [op2] + [CF]

Der Inhalt der beiden Operanden wird addiert und die Summe im linken Operanden abgelegt.

Ist das Uebertragsflag (CF) gesetzt, so wird zum Ergebnis zusaetzlich eine 1 addiert.

Es werden 3 Typen des Befehls **ADC** unterschieden:

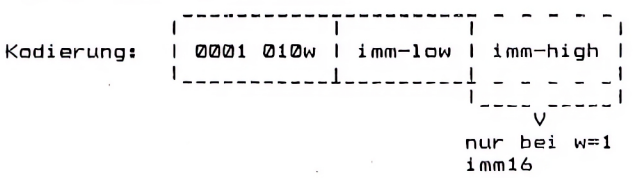
Typ 1: **[Speicher oder Register] + [Register]**



Beispiele:

```
ADC   AX,SI
ADC   CH,BL
ADC   BX,WORD PTR SUMM1
ADC   WORD PTR SUMM2[SI],CX
ADC   BYTE PTR BSUM[BX][DI],BL
```

Typ 2: **[Akku] + Direktwert**

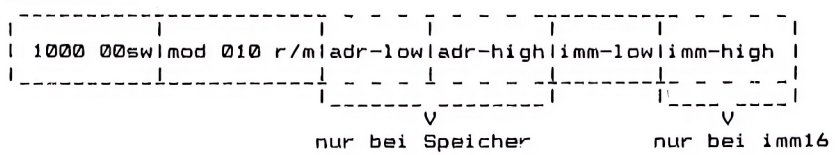


Beispiele:

```
ADC   AL,9
ADC   AX,7777
```

Typ 3: **[Speicher oder Register] + Direktwert**

Kodierung:



Beispiele:

```
ADC   BH,99
ADC   CX,6789
ADC   BYTE PTR SUMME[BP][DI],88
ADC   WORD PTR GESAMT[DI],4567H
ADC   SUMM,6271
```

4.6.3. Dekrementieren

Befehl:	DEC	Decrement destination by one
Schreibweise:	DEC op1	

Flags: OF, SF, ZF, AF, PF: X

Operandenkombinationen:

Typ	op1
1	reg16
2	reg8
	mem8
	mem16
	memi8
	memi16

Befehlswirkung: [op1] := [op1] - 1

Der Inhalt des Operanden wird um 1 dekrementiert.

Es werden 2 Typen des Befehls **DEC** unterschieden:

Typ 1: **Wortregister**

Kodierung:

0100 1 reg

Beispiele:

```
DEC   AX
DEC   BX
DEC   DI
```

Typ 2: **Speicher oder Byteregister**

Kodierung:

1111 111w	mod 001 r/m	adr-low	adr-high
		nur bei Speicher	

Beispiele:

```
DEC   AL
DEC   BH
DEC   BYTE PTR ZAHL
DEC   WORD PTR SPEICH[DI]
DEC   ALPHA[BX][SI]
```

4.6.4 Inkrementieren

Befehl:	INC	Increment destination by one
Schreibweise:	INC op1	

Flags: OF, SF, ZF, AF, PF: X

Operandenkombinationen:

Typ	op1
1	reg16
2	reg8 mem8 mem16 memi8 memi16

Befehlswirkung: [op1] := [op1] + 1

Der Inhalt des Operanden wird um 1 erhoeht.

Es werden 2 Typen des Befehls **INC** unterschieden:

Typ 1: **Wortregister**

Kodierung:

0100 0 reg

Beispiele:

```
INC   SI
INC   CX
INC   AX
```

Typ 2: **Speicher oder Byteregister**

Kodierung:

1111 111w	mod 101 r/m	adr-low	adr-high

nur bei Speicher

Beispiele:

```
INC   AL
INC   CH
INC   BYTE PTR ZAHL
INC   WORD PTR SPEICH[SI]
INC   ALPHA[BX][SI]
```

4.6.5. Subtraktion ohne Uebertrag

Befehl:	SUB	Subtraction
Schreibweise:	SUB op1,op2	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen:

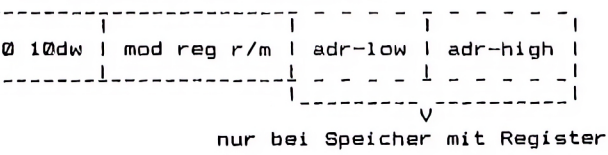
Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

Befehlswirkung: [op1] := [op1] - [op2]

SUB fuehrt eine Subtraktion des rechten Operanden vom linken Operanden durch und das Ergebnis wird im linken Operanden abgespeichert. Der rechte Operand bleibt erhalten.

Es werden 3 Typen des Befehls **SUB** unterschieden:

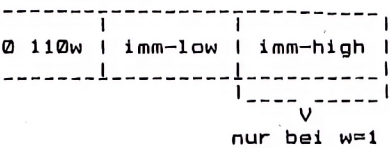
Typ 1: **[Speicher oder Register] mit [Register]**



Beispiele:

```
SUB   AX,BX
SUB   CH,DL
SUB   DI,ALPHA[SI]
SUB   BL,BYTE PTR MEMORY[BX][DI]
SUB   BYTE PTR MEM[DI],AL
SUB   WORD,BX
```

Typ 2: **[Akku] mit Direktwert**

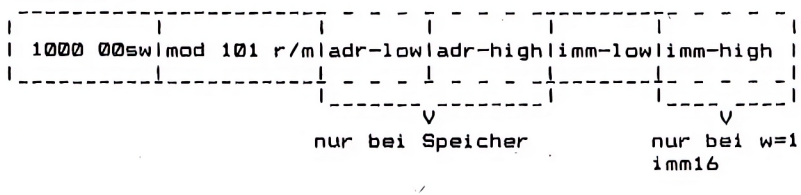


Beispiele:

```
SUB   AL,6
SUB   AX,300H
```

Typ 3: **[Speicher oder Register] mit Direktwert**

Kodierung:



Beispiele:

```
SUB   BX,2001
SUB   CL,0FFH
SUB   BYTE PTR MEMORY,12
SUB   MEM_WORD[BX],791
SUB   GAMMA[BX][DI],1987
```

4.6.6. Subtraktion mit Uebertrag

Befehl:	SBB	Subtraction with borrow
Schreibweise:	SBB op1,op2	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen:

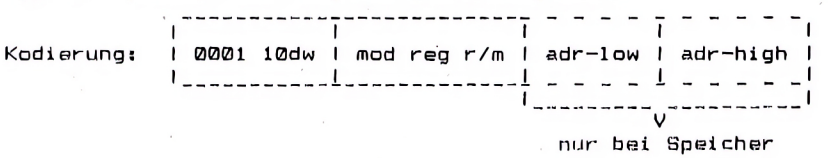
Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

Befehlswirkung: [op1] := [op1] - [op2] - [CF]

SBB fuehrt eine Subtraktion des rechten Operanden vom linken Operanden durch. Falls das Uebertragsflag (CF)=1 ist, wird vom Ergebnis noch eine 1 subtrahiert.

Es werden 3 Typen des Befehls **SBB** unterschieden:

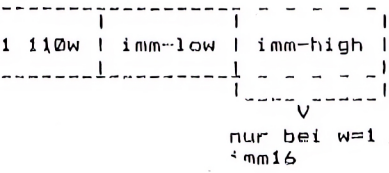
Typ 1: **[Speicher oder Register] mit [Register]**



Beispiele:

```
SBB   AX,CX
SBB   CL,DH
SBB   SI,BETA[DI]
SBB   CL,BYTE PTR MEMORY[BX][DI]
SBB   BYTE PTR MEM[SI],AL
SBB   WORD_1,BX
```

Typ 2: [Akku] mit Direktwert

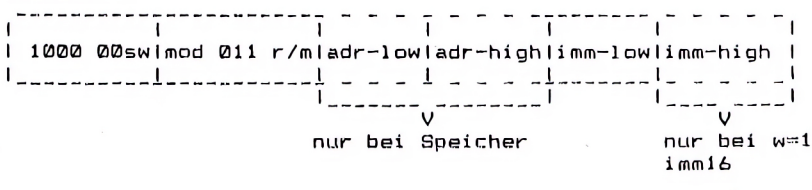


Beispiele:

```
SBB   AL,9
SBB   AX,409
```

Typ 3: [Speicher oder Register] mit Direktwert

Kodierung:



Beispiele:

```
SBB   BX,1000
SBB   CH,0FFH
SBB   BYTE PTR MEMORY,69
SBB   MEM_WORD[BX],788
SBB   GAMMA[BX][DI],1986
```

4.6.7. Multiplikation vorzeichenlos

Befehl:	MUL	Multiply accumulator by register or memory; unsigned
Schreibweise:	MUL op1	

Flags: OF, CF: X
SF, ZF, AF, PF: U

Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung: [AL|AX + ext8|ext16] := [AL|AX] * [op1]
IF [ext8|ext16] = 0 THEN [CF] = 0
ELSE [CF] = 1
[OF] := [CF]

MUL fuehrt eine vorzeichenlose Multiplikation mit dem Akkumulator und dem spezifizierten Operanden aus. Das Produkt wird im Akkumulator und seiner Erweiterung [ext8|ext16] abgespeichert. Dabei werden die Flags CF und OF gesetzt, wenn die obere Haelfte des Produkts (in der Erweiterung des Akkumulators) nicht Null ist, sonst sind die beiden Flags nicht gesetzt. Der Operand bleibt erhalten. In bestimmten Faellen sind mehrfache Additions- und Verschiebeoperationen geeigneter, um Multiplikationen auszufuehren. Dies besonders dann, wenn nur wenig Zeit zur Verfuegung steht und Speicherinhalte nicht erhalten bleiben muessen.

Kodierung:

1111 011w	mod 100 r/m	adr-low	adr-high

nur bei Speicher

Beispiele:

a)
```
MOV   AL,BYTE PTR FAKT1
MUL   BYTE PTR FAKT2
; Produkt in AX
```

b)
```
MOV   AX,FAKTOR1
MUL   FAKTOR2
; Produkt: - hoeherwertig in DX
;          - niederwertig in AX
```

4.6.8. Multiplikation vorzeichenbehaftet, ganzzahlig

Befehl:	IMUL	Integer multiply accumulator by register or memory; signed
Schreibweise:	IMUL op1	

Flags: OF, CF: X
SF, ZF, AF, PF: U

Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung:
```
[AL|AX + ext8|ext16] := [AL|AX] * [op1]
IF [ext8|ext16] = sign-extension of [low]
   THEN [CF] = 0
   ELSE [CF] = 1
[OF] := [CF]
```

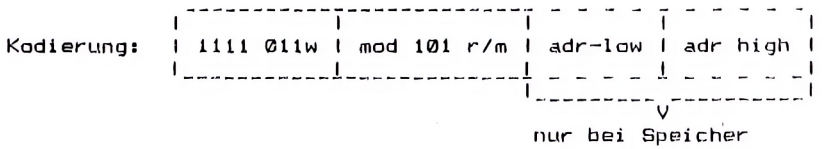
IMUL fuehrt eine ganzzahlige, vorzeichenbehaftete Multiplikation mit dem Akkumulator und dem spezifizierten Operanden aus (Interpretation als Binaerzahl mit Vorzeichen). Das Produkt wird im Akkumulator und seiner Erweiterung abgespeichert.

Der Operand bleibt erhalten.

Der Wert eines 8-Bit-Operanden (7 Bit plus Vorzeichen) liegt zwischen - 128 und + 127; der Wert eines 16-Bit-Operanden (15

Bit plus Vorzeichen) liegt zwischen - 32788 und + 32767.

Wenn die 8 bzw. 16 hoeherwertigen Bit (ext8 | ext16) des Produkts alle den Wert des Vorzeichens der niederwertigen Haelfte des Produkts haben, dann werden OF und CF auf Null gesetzt, andernfalls werden beide auf 1 gesetzt.



Beispiele:

```
a)  MOV   BYTE PTR FAKT1
    IMUL  FAKT2
    ; Produkt in AX

b)  MOV   AX,MEM_FAKT[DI]
    IMUL  FAKTOR2
    ; Produkt: - hoeherwertige Haelfte in DX
    ;          - niederwertige Haelfte in AX

c)  MOV   AL,BYTE PTR FAKT1
    CBW       ; Umwandlung Byte in Wort in AX
    IMUL  FAKTOR2
    ; Produkt: - hoeherwertig in DX
    ;          - niederwertig in AX
```

4.6.9. Division vorzeichenlos

Befehl:	DIV	Division unsigned
Schreibweise:	DIV opi	

Flags: OF, SF, ZF, AF, PF, CF: U

Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung: Quotient:
[AL|AX] := [AL|AX + ext8|ext16] / [op1]
Rest:
[ext8|ext16] := [AL|AX + ext8|ext16] mod [op1]
IF Quotient > FFH bei Byte oder
> FFFFH bei Wort (bei Division durch Null)
THEN Quotient und Rest undefiniert,
INT 0 ausgeloest

DIV fuehrt eine Division ohne Vorzeichen eines Dividenden mit der doppelten Laenge, der sich im Akkumulator und seiner Erweiterung befindet, und eines Divisors einfacher Laenge, der sich im spezifizierten Operanden befindet, durch.

Der Quotient einfacher Laenge wird im Akkumulator und der Rest in seiner Erweiterung abgelegt.

Bei Division durch Null sind Quotient und Rest undefiniert und der INT 0 wird ausgeloest.

Kodierung:

1111 011w	mod 110 r/m	adr-low	adr-high

nur bei Speicher

Beispiele:

```
; Division Wort durch Byte
MOV   AX,NUM_WORD
DIV   BYTE PTR DIVISOR  ; Quotient in AL, Rest in AH

; Division  Byte durch Byte
MOV   AL,NUM_BYTE
SUB   AH,AH             ; Wandeln Byte in AL zu Wort in AX
DIV   DIVISOR_BYTE      ; Quotient in AL, Rest in AH
```

```
; Division Doppelwort durch Wort
MOV   DX,HIGH_WORD
MOV   AX,LOW_WORD
DIV   DIVISOR_WORD       ; Quotient in AX, Rest in DX

;Division Wort durch Wort
MOV   AX,NUM_WORD
SUB   DX,DX              ; Wandlung Wort in Doppelwort
DIV   DIVISOR_WORD       ; Quotient in AX, Rest in DX
```

4.6.10. Division vorzeichenbehaftet, ganzzahlig

Befehl:	IDIV	Integer division; signed
Schreibweise:	IDIV op1	

Flags: OF, SF, ZF, AF, PF, CF: U

Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung:

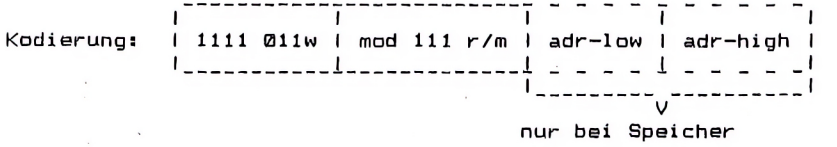
Quotient: [AX|AL] := [AL|AX + ext8|ext16] / [op1]
Rest: [ext8|ext16] := |[AL|AX + ext8|ext16] mod [op1]|
Voraussetzung: [AL|AX] $\geq$ 0
IF Quotient > 7FH bei Byte oder
7FFFH bei Wort (bei Division durch Null)
THEN Quotient und Rest undefiniert und Ausloesung von INT 0

IDIV fuehrt eine vorzeichenbehaftete Division eines Dividenden mit der doppelten Laenge, der sich im Akkumulator und seiner Erweiterung befindet, und eines Divisors einfacher Laenge, der sich im spezifizierten Operanden befindet, aus.

Der Quotient einfacher Laenge wird im Akkumulator und der Rest in seiner Erweiterung DX abgelegt. Der Operand bleibt erhalten.

Bei Division durch Null sind Resultat und Rest undefiniert und der INT 0 wird ausgeloest. Die Operanden werden als vorzeichen-

behaftete Binaerzahlen interpretiert.



Beispiele:

```
     ; Division Byte durch Byte

     MOV   AL,BYTE PTR NUM
     SUB   AH,AH                ; Wandel Byte in AL zu Wort in AX
     IDIV  BYTE PTR DIVISOR     ; Quotient in AL, Rest in AH

     ; Division Wort durch Wort

     MOV   AX,NUM_WORD
     SUB   DX,DX                ; Wandlung in Doppelwort
     IDIV  DIVISOR_WORD         ; Quotient in AX, Rest in DX
```

4.6.11. Leeroperation

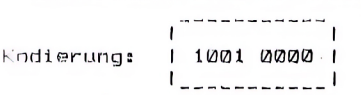
Befehl:	NOP	No operation
Schreibweise:	NOP	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung: Leeroperation

NOP fuehrt eine Leeroperation aus. Der Befehl benoetigt 3 Takte. Anschliessend wird der naechste Befehl ausgefuehrt.



Beispiel:

```
     NOP
```

4.6.12. Arithmetische Negation = Zweierkomplement

Befehl:	NEG	Negate, or form 2's complement
Schreibweise:	NEG op1	

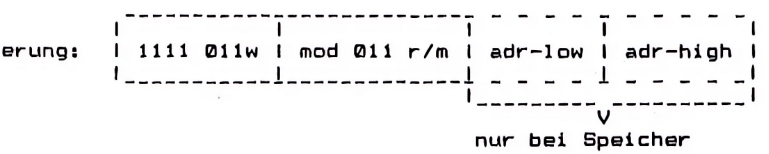
Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung: [op1] := 0 - [op1]
IF [op1] = 0 THEN [CF] = 0
ELSE [CF] = 1

NEG fuehrt eine Subtraktion des spezifizierten Operanden von Null durch und speichert das Ergebnis wieder im Operanden ab. Damit steht im Operanden nach der Operation das Zweierkomplement.



Beispiele:

```
NEG   AX
NEG   BYTE PTR MEM
NEG   AL
NEG   WORD PTR EMIL[BX][DI]
```

Wenn AL vor der Operation den Wert 13H (0001 0011B) enthielt, so lautet das Ergebnis nach der Operation -13H oder 0EDH (1110 1101B).

4.6.13. Umkodierung von AL

Befehl:	XLAT	XLATB	Table look-up translation
Schreibweise:	XLAT	XLATB	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung: [AL] := [DS:BX] + [AL]

Dieser Befehl wird verwendet, um in einer Kodewandeltabelle ein bestimmtes Zeichen zu adressieren.

Die Register AL und BX muessen vor der Operation eingestellt werden. In BX muss der Offset des Tabellenanfangs im aktuellen Datensegment (Tabelle max. 256 Bytes gross) geladen werden.

In AL muss der Index [[BX] + [AL]] des Elements, bezueglich zum Tabellenanfang, dessen Wert aus der Tabelle geholt werden soll, eingetragen werden.

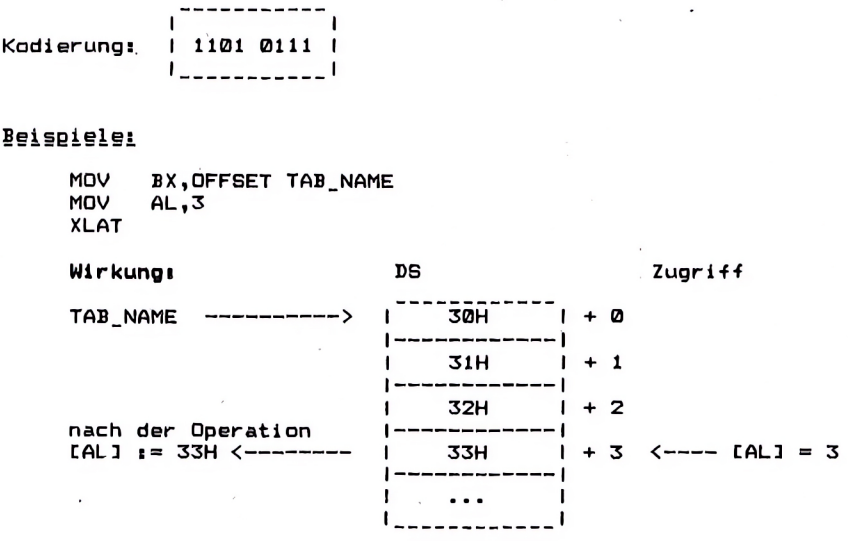
Die Tabelle muss vorher im Datensegment definiert sein. Mit der Operation **XLAT** wird das entsprechende Byte aus der Tabelle geholt und in AL eingetragen.

Kodierung:

1101 0111

Beispiele:

```
MOV   BX,OFFSET TAB_NAME
MOV   AL,3
XLAT
```



4.6.14. Vergleich

Befehl:	CMP	Compare two operands
Schreibweise:	CMP op1,op2	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	reg8	{mem8|memi8}
	{mem8|memi8}	reg8
	reg16	{mem16|memi16}
	{mem16|memi16}	reg16
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	{mem8|memi8}	imm8
	{mem16|memi16}	imm16

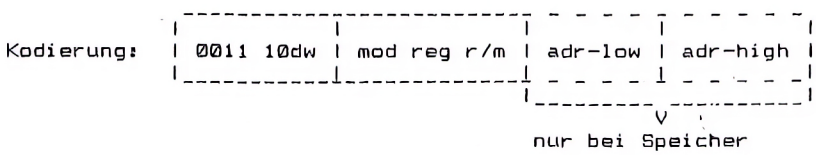
Befehlswirkung: Vergleich [op1] mit [op2]

Der rechte Operand [op2] wird vom linken Operanden [op1] subtrahiert. Dabei werden nur die Flags eingestellt. Die beiden Operanden bleiben unveraendert und muessen vom gleichen Typ (Byte oder Wort) sein.

Ausnahme ist der Vergleich eines Direktwertbytes mit einem Speicherwort.

Es werden 3 Typen des Befehl **CMP** unterschieden:

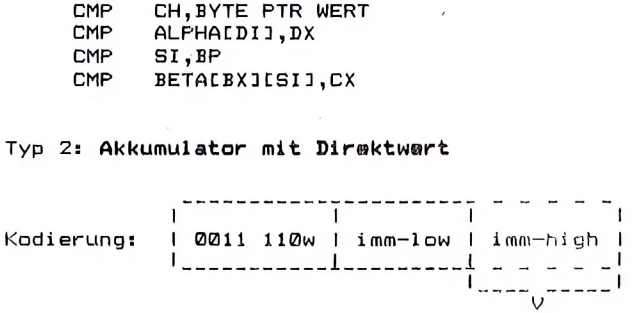
Typ 1: **Register oder Speicher mit Register**



Beispiele:

```
CMP   BH,CL
CMP   BX,DX
CMP   CH,BYTE PTR WERT
CMP   ALPHA[DI],DX
CMP   SI,BP
CMP   BETA[BX][SI],CX
```

Typ 2: **Akkumulator mit Direktwert**

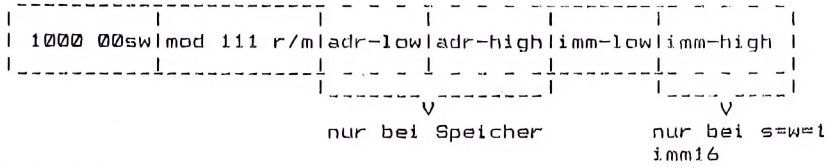


Beispiele:

```
CMP   AL,6
CMP   AX,999
CMP   AX,0FFFFH
CMP   AL,0FFH
```

Typ 3: **Register oder Speicher mit Direktwert**

Kodierung:



Beispiele:

```
CMP   WORD PTR [BX][DI],6ACEH
CMP   GAMMA[BX],67
CMP   SI,798
CMP   BH,7
CMP   DX,0FFFFH
```

4.7. Logikbefehle

4.7.1. Logisches UND

Befehl:	AND	Logical AND
Schreibweise:	AND op1,op2	

Flags: SF, ZF, PF: X
OF, CF : 0
AF : U

Operandenkombinationen:

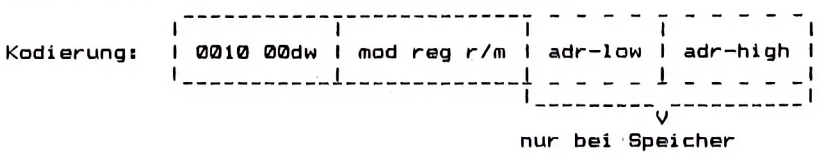
Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

Befehlswirkung: [op1] := [op1] & [op2]
[CF] := 0
[OF] := 0

Mit **AND** werden die beiden Operanden bitweise logisch verknuepft. Das Ergebnis ist nur in solchen Bits gleich 1, wo beide Operanden eine 1 in diesem Bit stehen hatten, sonst ist es 0. Das Resultat wird im linken Operanden [op1] abgespeichert, der rechte Operand [op2] bleibt erhalten. Diese Operation setzt die Flags OF und CF auf Null.

Es werden 3 Typen des Befehls **AND** unterschieden:

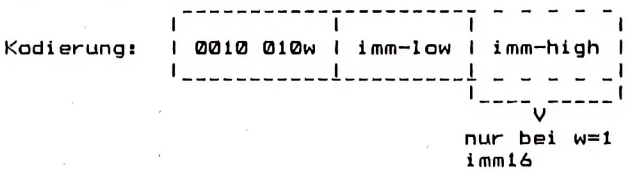
Typ 1: **Register oder Speicher mit Register**



Beispiele:

```
AND   AL,BH
AND   AX,BX
AND   DH,BYTE PTR MEMORY
AND   ALPHA[DI],AX
AND   DX,BETA[BX][SI]
```

Typ 2: **Akku mit Direktwert**

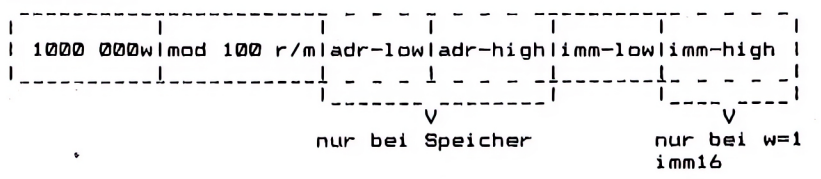


Beispiele:

```
AND   AX,2323H
AND   AL,39H
AND   AL,26
AND   AX,679
```

Typ 3: **Register oder Speicher mit Direktwert**

Kodierung:



Beispiele:

```
AND   BL,7AH
AND   DX,0FFFFH
AND   BYTE PTR BEIM[BX][SI],10011111B
AND   MEM_WORD,6060H
AND   ALPHA[DI],2323H
```

4.7.2. Logisches ODER

Befehl:	OR	Logical OR
Schreibweise:	OR op1,op2	

Flags: SF, ZF, PF: X
OF, CF : 0
AF : U

Operandenkombinationen:

Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	{mem8|memi8}	reg8
	{mem16|memi16}	reg16
	reg8	{mem8|memi8}
	reg16	{mem16|memi16}
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	{mem8|memi8}	imm8
	{mem16|memi16}	imm16

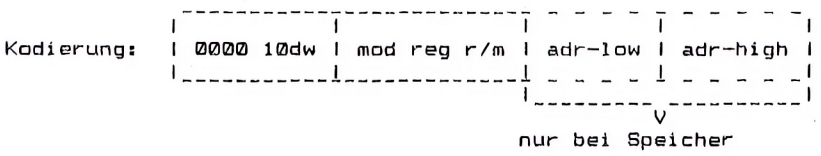
Befehlswirkung: [op1] := [op1] | [op2]
[CF] := 0
[OF] := 0

Mit OR werden die beiden Operanden bitweise logisch verknuepft. Das Ergebnis ist nur in solchen Bitpositionen gleich Null, wo beide Operanden eine 0 besitzen; sonst ist es 1.

Das Resultat wird im linken Operanden [op1] abgespeichert, der rechte Operand [op2] bleibt erhalten. Die Flags OF und CF werden geloescht.

Es werden 3 verschiedene Typen des Befehls OR unterschieden:

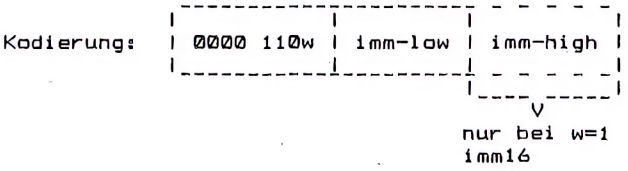
Typ 1: **Register oder Speicher mit Register**



Beispiele:

```
OR    AL,CH
OR    BX,DX
OR    CL,BYTE PTR MEMORY
OR    BETA[SI],AX
OR    DX,GAMMA[BX][SI]
```

Typ 2: **Akku mit Direktwert**

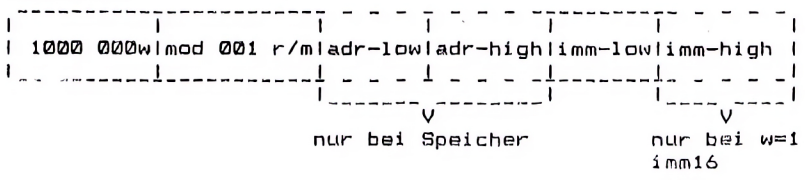


Beispiele:

```
OR    AX,3030H
OR    AL,31h
OR    AL,100
OR    AX,512
```

Typ 3: **Register oder Speicher mit Direktwert**

Kodierung:



Beispiele:

```
OR    BH,5AH
OR    DX,2340H
OR    BYTE PTR WERT[BX][DI],60H
OR    MEM_WORD,5050H
OR    PLATZ[DI],3030H
```

4.7.3. Exklusiv - ODER

Befehl:	XOR	Logical Exklusive OR
Schreibweise:	XOR op1,op2	

Flags: SF, ZF, PF: X
OF,CF : 0
AF : U

Operandenkombinationen:

Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

Befehlswirkung: [op1] := [op1] || [op2]
[CF] := 0
[OF] := 0

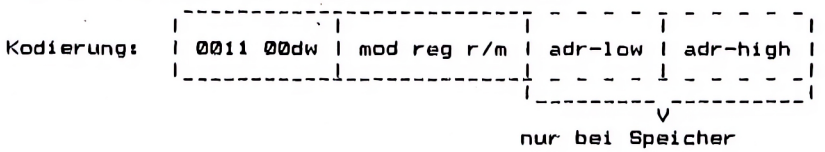
Mit **XOR** werden die beiden Operanden bitweise logisch verknuepft. Das Ergebnis ist nur in den Bitpositionen gleich Null, wo die entsprechenden Bits beider Operanden gleich sind, sonst ist es 1.

Das Resultat wird im linken Operanden [op1] abgespeichert, der rechte Operand [op2] bleibt erhalten. Die Flags OF und CF werden

geloescht.

Es werden 3 Typen des Befehls **XOR** unterschieden:

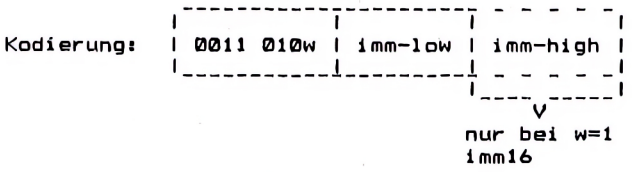
Typ 1: **Register oder Speicher mit Register**



Beispiele:

```
XOR   AL,AL    ; AL loeschen
XOR   BX,BX    ; BX loeschen
XOR   CL,BYTE PTR MEMORY
XOR   BETA[SI],AX
XOR   DX,GAMMA[BX][SI]
```

Typ 2: **Akku mit Direktwert**

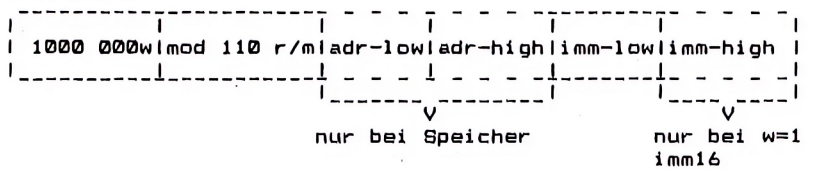


Beispiele:

```
XOR   AX,3030H
XOR   AL,30H
XOR   AL,100
XOR   AX,1024
```

Typ 3: **Register oder Speicher mit Direktwert**

Kodierung:



Beispiele:

```
XOR   BH,5AH
XOR   DX,3040H
XOR   BYTE PTR WERT[BX][DI],30H
XOR   MEM_WORD,6060H
XOR   PLATZ[SI],3030H
```

4.7.4. Logisches UND ohne Resultatsabspeicherung

Befehl:	TEST	Test, or logical compare
Schreibweise:	TEST op1,op2	

Flags: SF, ZF, PF: X
OF, CF : 0
AF : U

Operandenkombinationen:

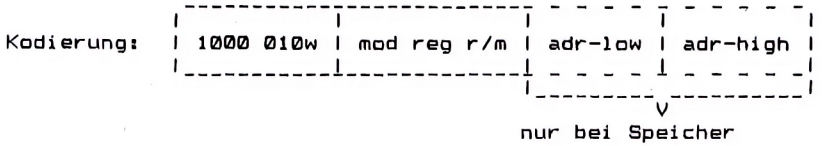
Typ	op1	op2
1	reg8	reg8
	reg16	reg16
	(mem8|memi8)	reg8
	(mem16|memi16)	reg16
	reg8	(mem8|memi8)
	reg16	(mem16|memi16)
2	AL	imm8
	AX	imm16
3	reg8	imm8
	reg16	imm16
	(mem8|memi8)	imm8
	(mem16|memi16)	imm16

Befehlswirkung: [op1] & [op2]
[CF] := 0
[OF] := 0

Mit **TEST** werden die Operanden mit dem logischen UND bitweise verknuepft, ohne das sie dabei veraendert werden. Es erfolgt nur eine Einstellung der Flags. Die Flags CF und OF werden geloescht.

Es werden 3 Typen des Befehls **TEST** unterschieden:

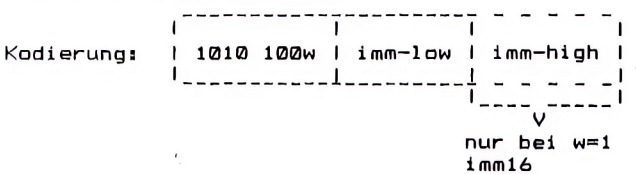
Typ 1: **Register oder Speicher mit Register**



Beispiele:

```
TEST    AL,CH
TEST    BX,DX
TEST    CL,BYTE PTR MEMORY
TEST    BETA[SI],AX
TEST    DX,GAMMA[BX][SI]
```

Typ 2: **Akku mit Direktwert**

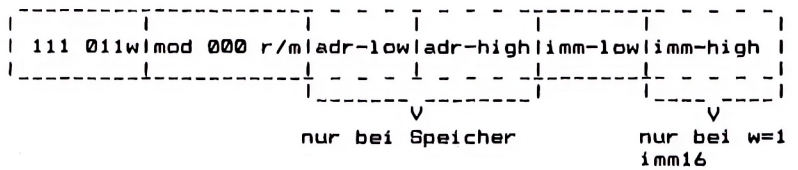


Beispiele:

```
TEST    AX,3030H
TEST    AL,30H
TEST    AL,100
TEST    AX,1024
```

Typ 3: **Register oder Speicher mit Direktwert**

Kodierung:



Beispiele:

```
TEST    BH,5AH
TEST    DX,3040H
TEST    BYTE PTR WERT[BX][DI],30H
TEST    MEM_WORD,6060H
TEST    PLATZ[SI],3030H
```

4.7.5. Logisches Komplement

Befehl:	NOT	Not, or form 1's complement
Schreibweise:	NOT op1	

Flags: keine Beeinflussung

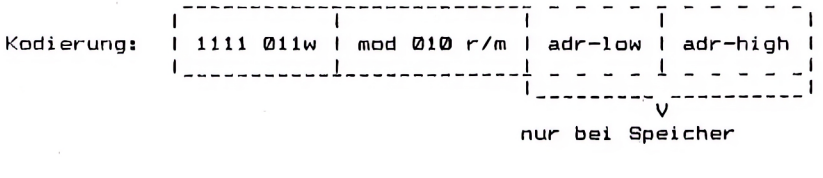
Operandenkombinationen:

Typ	op1
1	reg8
	reg16
	mem8
	mem16
	memi8
	memi16

Befehlswirkung:

Negation bzw. 1-er Komplement
- bei Byteoperationen: [op1] := 0FFh - [op1]
- bei Wortoperationen: [op1] := 0FFFFH - [op1]

Der spezifizierte Operand wird von 0FFH (Byte) oder 0FFFFH (Wort) subtrahiert und das Resultat wird im Operanden abgespeichert.



Beispiele:

```
NOT   AH        ; falls AH = 13H, dann Resultat = ØECH
NOT   MEM_WORD  ; 2FC3H ---> ØD3CH
```

4.8. Sprungbefehle

Es werden hierbei bedingte Sprungbefehle mit kurzer Distanz und unbedingte Sprungbefehle unterschieden.

4.8.1. Bedingte Sprungbefehle mit kurzer Distanz

Befehle:	Jcond	Jump on condition
Schreibweise:	Jcond op1	

Flags: keine Beeinflussung

Operandenkombinationen: sdisp

Befehlswirkung: IF cond = wahr, THEN IP := IP + sdisp (Jump)
ELSE no Jump

Diese Spruenge werden ausgefuehrt, wenn die im Befehl angegebene Bedingung wahr ist, sonst erfolgt kein Sprung.

Der Distanzbereich, in dem diese Spruenge ausgefuehrt werden koennen, betraegt $-128 \leq sdisp \leq +127$ Bytes.

Kodierung:

Opcode	sdisp

Beispiele:

```
a)          MOV   CX,5
             .
     MARKE:  .
             .
             JCXZ  MARK1
             DEC   CX
             JMP   MARKE
              .
     MARK1:   .

b)          JNZ   $+98
```

```
c)          CMP   CX,DX
            JE    LAB2
            INC   CX
             .
      LAB2:  .

d)          SUB   CX,DX
            JZ    LAB3
            INC   CX
             .
      LAB3:  .
```

Tabelle aller bedingten Sprungbefehle

Bedingung (cond)	mathem.	Flag	Werte vorzeichenhaftet
JA JNBE	> not ≤	CF = 0 & ZF = 0	ohne Vz
JAE JNB JNC	≥ not <	CF = 0	ohne Vz
JB JNAE JC	< not ≥	CF = 1	ohne Vz
JBE JNA	≤ not >	CF = 1 | ZF = 1	ohne Vz
JE JZ	=	ZF = 1	-
JNE JNZ	≠	ZF = 0	-
JG JNLE	> not ≤	OF = SF & ZF = 0	mit Vz
JGE JNL	≥ not <	OF = SF	mit Vz
JL JNGE	< not ≥	OF ≠ SF	mit Vz
JLE JNG	≤ not >	OF ≠ SF | ZF = 1	mit Vz
JS	Vz negativ	SF = 1	mit Vz
JNS	Vz positiv	SF = 0	mit Vz
JPO JNP	ungerade Paritaet	PF = 0	-
JPE JP	gerade Paritaet	PF = 1	-
JNO	kein Ueberl	OF = 0	-
JO	Ueberlauf	OF = 1	-
JCXZ	[CX] = 0	-	-

4.8.2. Unbedingter Sprung

Befehl:	JMP	Jump
Schreibweise:	JMP op1	

Flags: keine Beeinflussung

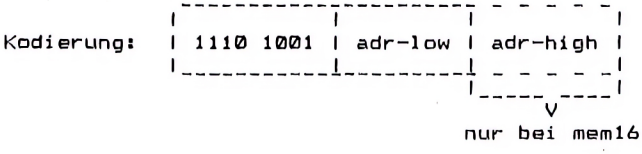
Operandenkombinationen:

Typ	op1
1	mem16 SHORT mem16
2	reg16 memi16
3	FAR mem16
4	memi16

Befehlswirkung: Es wird unbedingt an die im Operanden angegebene Stelle im Programm verzweigt.

Es werden 4 Typen des Befehls **JUMP** unterschieden:

Typ 1: **Im Segment oder Gruppe, direkt**



Operandenkombinationen: mem16
SHORT mem16 (Sprung auf Kurzdistanz, Befehl nur 2 Byte)

Beispiel:

```
JMP   MARK1  ; Es wird zur Marke MARK1 im gleichen
             ; Kodesegment gesprungen.
```

Typ 2: **Im Segment oder Gruppe, indirekt**

Operandenkombinationen: reg16
memi16

Wird ein 16-Bitregister (reg16) adressiert, dann wird der Inhalt dieses Registers in den Befehlszeiger (IP) uebertragen. Wird ein Speicherplatz (mem16) adressiert, so wird der Inhalt dieses und des darauffolgenden Speicherplatzes in den Befehlszeiger (IP) uebernommen.

Der Inhalt des Kodesegmentregisters (CS) wird nicht veraendert. Anschliessend wird der Befehl ausgefuehrt, dessen 1. Byte durch den neuen Inhalt des IP in Verbindung mit dem Inhalt des Registers CS adressiert wird.

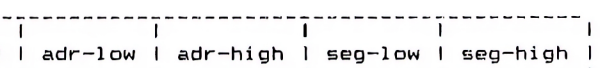
Beispiele:

```
a) ; mit reg16
   JMP   AX ; Es wird zur im Register AX stehenden Adresse
            ; verzweigt.

b) ; mit memi16
   JMP   ALPHA[BX]
   JMP   WORD PTR MARK[BX][SI]
```

Typ 3: **Inter-Segment** (in ein anderes Kodesegment), **direkt**

Kodierung:



Operandenkombination: FAR mem16

Beispiel:

```
JMP   FAR MARKE
```

Das Praefix **FAR** muss als Kennzeichnung fuer einen Sprung ueber die Segmentgrenze hinaus geschrieben werden.

Typ 4: **Inter-Segment, indirekt**

Operandenkombination: memi16

Der Inhalt des durch den Befehl adressierten Wortes wird in den Befehlszeiger (IP), der Inhalt des darauffolgenden Wortes wird in das Kodessegmentregister (CS) uebertragen. Anschliessend wird der Befehl ausgefuehrt, dessen absolute Adresse sich aus den neuen Inhalten von CS und IP ergibt.

Beispiel: JMP DWORD PTR [BX][SI]

4.9. Verschiebe- und Rotationsbefehle

4.9.1. Verschiebebefehle

4.9.1.1. Arithmetische oder logische Linksverschiebung

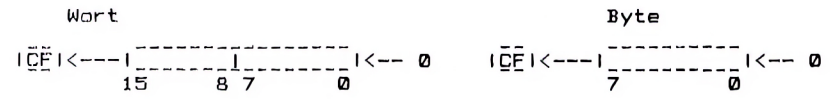
Befehl:	arithm.: SAL logisch: SHL	Shift arithmetic left Shift logical left
Schreibweise:	SAL op1,op2 SHL op1,op2	

Flags: OF, SF, ZF, PF, CF: X
AF : U

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	(mem8|memi8)	1
	(mem16|memi16)	1
2	reg8	CL
	reg16	CL
	(mem8|memi8)	CL
	(mem16|memi16)	CL

Befehlswirkung:



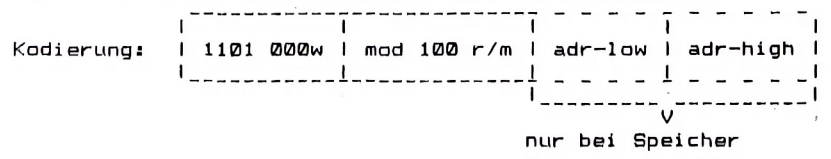
SAL und **SHR** verschieben den linken Operanden um die im rechten Operanden stehende Anzahl von Bits nach links. Dabei werden von rechts die Bits mit dem Wert Null aufgefuellt.

SAL - Arithmetische Linksverschiebung
SHL - Logische Linksverschiebung

Beide Befehle fuehren die gleiche Operation aus.

Es werden zwei Typen dieser Befehle unterschieden:

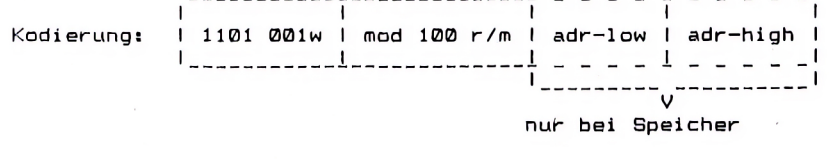
Typ 1: **[Register oder Speicher] um 1 Bit verschieben**



Beispiele:

```
SAL   AL,1
SAL   BH,1
SAL   BYTE PTR WERT[DI],1
SHL   MEM_WORD,1
SHL   AX,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL verschieben**



Beispiele:

```
MOV   CL,3 ; Einstellung Register CL
SAL   AL,CL  ; Verschiebung um 3 Bits nach links
SHL   BH,CL
SAL   AX,CL
SAL   BYTE PTR WERT[BX][DI],CL
SHL   MEM_WORD,CL
```

4.7.1.2. Arithmetische Rechtsverschiebung

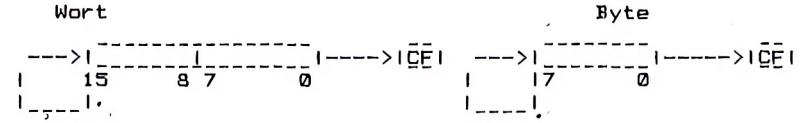
Befehl:	SAR	Shift arithmetic right
Schreibweise:	SAR op1,op2	

Flags: OF, SF, ZF, PF, CF: X
AF : U

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	(mem8|memi8)	1
	(mem16|memi16)	1
2	reg8	CL
	reg16	CL
	(mem8|memi8)	CL
	(mem16|memi16)	CL

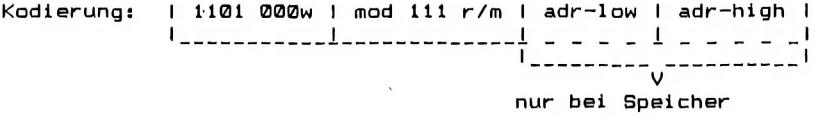
Befehlswirkung:



SAR verschiebt den linken Operanden um die im rechten Operanden stehende Anzahl von Bits nach rechts. Dabei ist die Verschiebung im hoechsten Bit gleich dem Originalbit an dieser Stelle vor der Verschiebung (wichtig fuer vorzeichenbehaftete Werte !).

Es werden 2 Typen des Befehls **SAR** unterschieden:

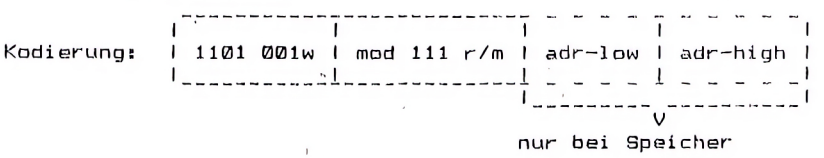
Typ 1: **[Register oder Speicher] um 1 Bit verschieben**



Beispiele:

```
SAR   AL,1
SAR   BH,1
SAR   BYTE PTR WERT[SI],1
SAR   MEM_WORD,1
SAR   AX,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL verschieben**



Beispiele:

```
MOV   CL,3  ; Einstellung Register CL
SAR   AL,CL ; Verschiebung um 3 Bits nach rechts
SAR   BH,CL
SAR   AX,CL
SAR   BYTE PTR WERT[BX][DI],CL
SAR   MEM_WORD,CL
```

4.7.1.3. Logische Rechtsverschiebung

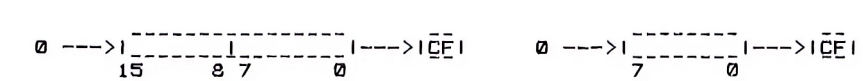
Befehl:	SHR	Shift logical right
Schreibweise:	SHR op1,op2	

Flags: OF, SF, ZF, PF, CF: X
AF : U

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	{mem8|memi8}	1
	{mem16|memi16}	1
2	reg8	CL
	reg16	CL
	{mem8|memi8}	CL
	{mem16|memi16}	CL

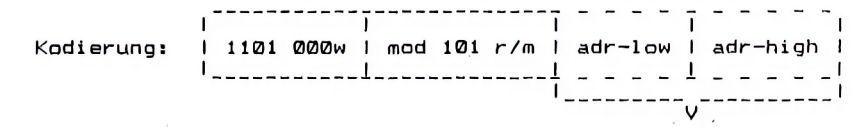
Befehlswirkung:



SHR verschiebt den linken Operanden um die Anzahl Bits, die im rechten Operanden spezifiziert worden sind, nach rechts. In das hoechstwertige Bit wird eine Null eingeschoben und das niederwertigste Bit wird in das Uebertragsflag (CF) verschoben.

Es werden 2 Typen des Befehls **SHR** unterschieden:

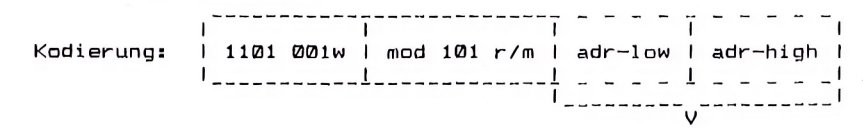
Typ 1: **[Register oder Speicher] um 1 Bit verschieben**



Beispiele:

```
SHR   AL,1
SHR   BH,1
SHR   BYTE PTR WERT[DI],1
SHR   MEM_WORD,1
SHR   AX,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL verschieben**



Beispiele:

```
MOV   CL,3  ; Einstellung Register CL
SHR   AL,CL ; Verschiebung um 3 Bits nach rechts
SHR   BH,CL
SHR   AX,CL
SHR   BYTE PTR WERT[BX][DI],CL
SHR   MEM_WORD,CL
```

4.9.2. Rotationsbefehle

4.9.2.1. Linksrotation

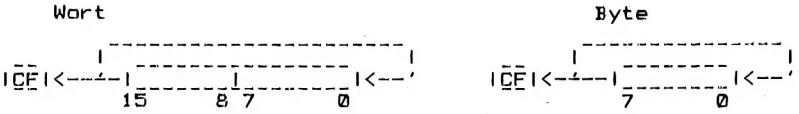
Befehl:	ROL	Rotate left
Schreibweise:	ROL op1,op2	

Flags: OF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	{mem8|memi8}	1
	{mem16|memi16}	1
2	reg8	CL
	reg16	CL
	{mem8|memi8}	CL
	{mem16|memi16}	CL

Befehlswirkung:

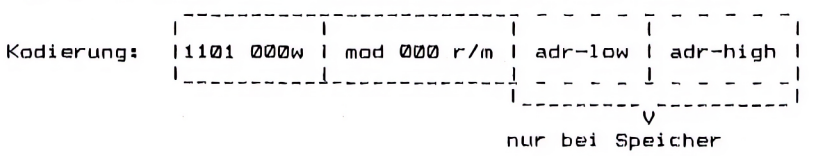


ROL rotiert den linken Operanden um die im rechten Operanden angegebene Zahl Bits. Wenn die Zahl der Bits im rechten Operanden 1 war und der neue Wert von CF nicht gleich dem neuen Wert des Bits hoechster Ordnung ist, dann ist das Flag OF gesetzt, sonst hat OF den Wert Null.

War die Zahl der Bits groesser 1, dann ist OF undefiniert.

Es werden 2 Typen des Befehls **ROL** unterschieden:

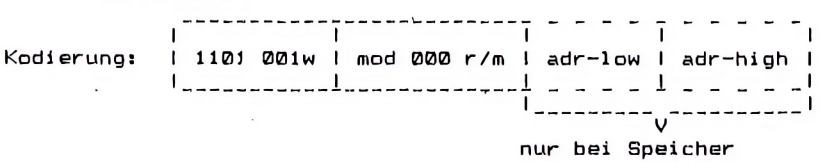
Typ 1: **[Register oder Speicherplatz] um 1 Bit rotieren**



Beispiele:

```
ROL   AL,1
ROL   BH,1
ROL   AX,1
ROL   BYTE PTR WERT[DI],1
ROL   MEM_WORD,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL rotieren**



Beispiele:

```
MOV   CL,4  ; Einstellung Register CL
ROL   AL,CL ; Rotation um 4 Bits nach links
ROL   BH,CL
ROL   AX,CL
ROL   BYTE PTR WERT[BX][DI],CL
ROL   MEM_WORD,CL
```

4.9.2.2. Rechtsrotation

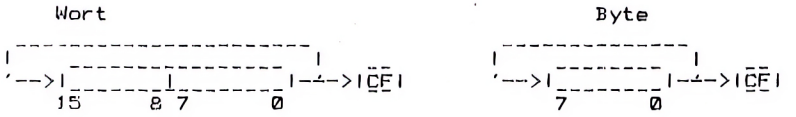
Befehl:	ROR	Rotate right
Schreibweise:	ROR op1,op2	

Flags: OF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	{mem8|memi8}	1
	{mem16|memi16}	1
2	reg8	CL
	reg16	CL
	{mem8|memi8}	CL
	{mem16|memi16}	CL

Befehlswirkung:

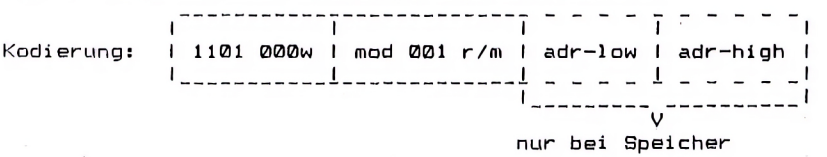


ROR rotiert den linken Operanden um die im rechten Operanden angegebene Zahl Bits.

Wenn die Zahl der Bits im rechten Operanden 1 war und der neue Wert von CF nicht gleich dem neuen Wert des Bits hoechster Ordnung ist, dann ist das O-Flag gesetzt, sonst hat OF den Wert Null. War die Zahl der Bits groesser 1, dann ist OF undefiniert.

Es werden 2 Typen des Befehls **ROR** unterschieden:

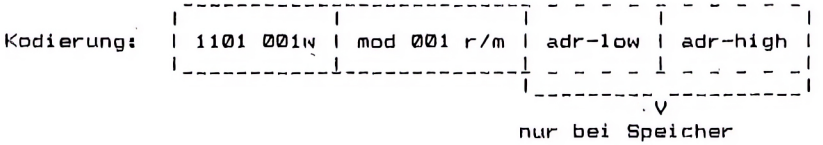
Typ 1: **[Register oder Speicher] um 1 Bit rotieren**



Beispiele:

```
ROR   AL,1
ROR   BH,1
ROR   AX,1
ROR   BYTE PTR WERT[DI],1
ROR   MEM_WORD,1
```

Typ 2: [Register oder Speicher] um die Zahl im Register CL rotieren



Beispiele:

```
MOV   CL,2  ; Einstellung Register CL
ROR   AL,CL ; Rotation um 2 Bits nach rechts
ROR   BH,CL
ROR   AX,CL
ROR   BYTE PTR WERT[BX][DI],CL
ROR   MEM_WORD,CL
```

4.9.2.3. Linksrotation durch CF

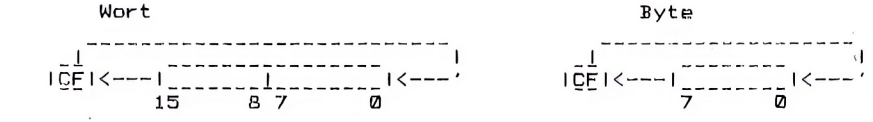
Befehl:	RCL	Rotate left through carry
Schreibweise:	RCL op1,op2	

Flags: OF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	(mem8|memi8)	1
	(mem16|memi16)	1
2	reg8	CL
	reg16	CL
	(mem8|memi8)	CL
	(mem16|memi16)	CL

Befehlswirkung:



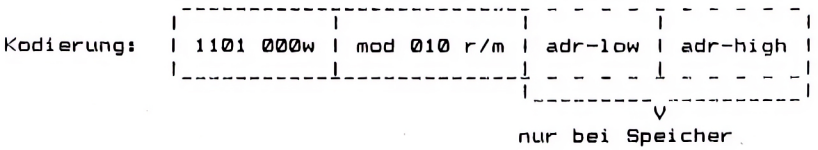
RCL rotiert den linken Operanden durch das Uebertragsflag (CF) um soviel Bits, wie im rechten Operanden angegeben wurde. Der zweite Operand kann die absolute Zahl 1 sein, oder die Zahl muss im Register CL stehen, wenn die Rotation um mehr als 1 Bit durchgefuehrt werden soll.

War der rechte Operand 1 und die zwei hoechsten Bits des zu rotierenden linken Operanden waren am Anfang ungleich (0 und 1 bzw. 1 und 0), dann wird das OF-Flag gesetzt. Sonst erhaelt OF den Wert Null.

War die Zahl im rechten Operanden nicht 1, dann ist OF undefiniert.

Es werden 2 Typen des Befehls **RCL** unterschieden:

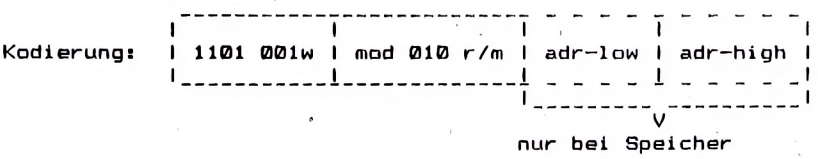
Typ 1: **[Register oder Speicher] um 1 Bit rotieren**



Beispiele:

```
RCL   AL,1
RCL   BH,1
RCL   AX,1
RCL   BYTE PTR WERT[DI],1
RCL   MEM_WORD,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL rotieren**



Beispiele:

```
MOV   CL,5  ; Einstellen Register CL
RCL   AL,CL ; Rotation um 5 Bits nach links
RCL   BH,CL
RCL   AX,CL
RCL   BYTE PTR WERT[BX][DI],CL
RCL   MEM_WORD,CL
```

4.9.2.4. Rechtsrotation durch CF

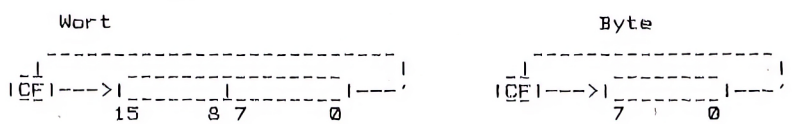
Befehl:	RCR	Rotate right through carry
Schreibweise:	RCR op1,op2	

Flags: OF, CF: X

Operandenkombinationen:

Typ	op1	op2
1	reg8	1
	reg16	1
	(mem8|memi8)	1
	(mem16|memi16)	1
2	reg8	CL
	reg16	CL
	(mem8|memi8)	CL
	(mem16|memi16)	CL

Befehlswirkung:

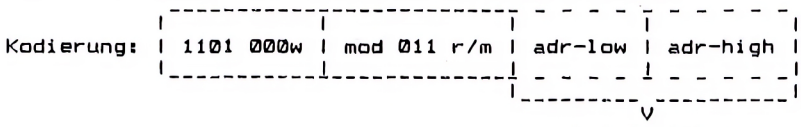


RCR rotiert den linken Operanden durch das Uebertragsflag (CF) um soviel Bits, wie im rechten Operanden angegeben wurde.

War der rechte Operand 1 und die zwei hoechsten Bits des zu rotierenden linken Operanden waren am Anfang ungleich (0 und 1 bzw. 1 und 0), dann wird das O-Flag gesetzt. Sonst enthaelt OF den Wert Null. War die Zahl im rechten Operanden nicht 1, dann ist OF undefiniert.

Es werden 2 Typen des Befehls **RCR** unterschieden:

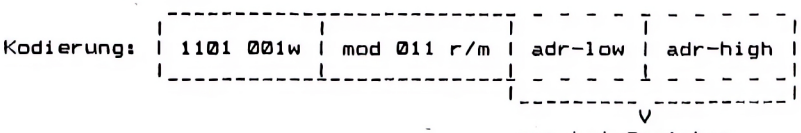
Typ 1: **[Register oder Speicher] um 1 Bit rotieren:**



Beispiele:

```
RCR   AL,1
RCR   BH,1
RCR   AX,1
RCR   BYTE PTR WERT[DI],1
RCR   MEM_WORD,1
```

Typ 2: **[Register oder Speicher] um die Zahl im Register CL rotieren**



Beispiele:

```
MOV   CL,4          ; Einstellen Register CL
RCR   AL,CL         ; Rotation um 4 Bits nach rechts
RCR   BH,CL
RCR   AX,CL
RCR   BYTE PTR WERT[BX][DI],CL
RCR   MEM_WORD,CL
```

4.10. Flagbefehle

4.10.1. Uebertragsflag setzen

Befehl:	STC	Set carry flag
Schreibweise:	STC	

Flags: CF: 1

Operandenkombinationen: keine Operanden

Befehlswirkung: [CF] := 1

STC setzt das Uebertragsflag (Carry-Flag).

Kodierung:

1111 1001

4.10.2. Uebertragsflag zuecksetzen

Befehl:	CLC	Clear carry flag
Schreibweise:	CLC	

Flags: CF: 0

Operandenkombinationen: keine Operanden

Befehlswirkung: [CF] := 0

CLC loescht das Uebertragsflag.

Kodierung:

1111 1000

4.10.3. Richtungsflag setzen

Befehl:	STD	Set direction flag
Schreibweise:	STD	

Flags: DF: 1

Operandenkombinationen: keine Operanden

Befehlswirkung: [DF] := 1

STD setzt das Richtungsflag DF (wichtig fuer Arbeit mit Zeichenkettenbefehlen).

Kodierung:

1111 1101

4.10.4. Richtungsflag ruecksetzen

Befehl:	CLD	Clear direction flag
Schreibweise:	CLD	

Flags: DF: 0

Operandenkombinationen: keine Operanden

Befehlswirkung: [DF] := 0

CLD loescht das Richtungsflag DF.

Kodierung:	1111 1100

4.10.5. Interruptflag setzen

Befehl:	STI	Set interrupt flag
Schreibweise:	STI	

Flags: IF: 1

Operandenkombinationen: keine Operanden

Befehlswirkung: [IF] := 1

STI setzt das Interruptflag IF und erlaubt maskierte externe Interrupts nach der Ausfuehrung des naechsten Befehls.

Kodierung:	1111 1011

4.10.6. Interruptflag ruecksetzen

Befehl:	CLI	Clear interrupt flag
Schreibweise:	CLI	

Flags: IF: 0

Operandenkombinationen: keine Operanden

Befehlswirkung: [IF] := 0

CLI loescht das Interruptflag IF. Damit werden externe Interrupts gesperrt.

Kodierung:	1111 1010

4.10.7. Flags in das Register AH laden

Befehl:	LAHF	Load AH from flags
Schreibweise:	LAHF	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:

LAHF transportiert die Flags SF, ZF, AF und CF in folgende Bits des Registers AH:

```
SF ----> Bit 7
ZF ----> Bit 6
AF ----> Bit 4
PF ----> Bit 2
CF ----> Bit 0
```

Die Bits 1, 3 und 5 des Registers AH sind undefiniert, d.h., sie koennen die Werte 0 oder 1 haben.

Kodierung:	1001 1111

4.10.8. Flags von AH in das Flagregister laden

Befehl:	SAHF	Store AH in flags
Schreibweise:	SAHF	

Flags: SF, ZF, AF, PF, CF: X

Operandenkombinationen: keine Operanden

Befehlswirkung:

SAHF transportiert die entsprechenden Bits des Registers AH zu den Flags SF, ZF, AF, PF und CF. Die anderen Bits von AH werden ignoriert.

Folgende Bits von AH werden transportiert:

```
Bit 7 ----> SF
Bit 6 ----> ZF
Bit 4 ----> AF
Bit 2 ----> PF
Bit 0 ----> CF
```

Kodierung:	1001 1110

4.10.9. Negieren Uebertragsflag

Befehl:	CMC	Complement carry flag
Schreibweise:	CMC	

Flags: CF: X

Operandenkombinationen: keine Operanden

Befehlswirkung: IF [CF] = 0 THEN [CF] := 1
ELSE [CF] := 0

CMC negiert den Inhalt des Uebertragsflags CF.

Kodierung:	1111 0101

4.11. Unterprogrammaufruf und -rueckkehr

4.11.1. Unterprogrammaufruf

Befehl:	CALL	Call a procedure
Schreibweise:	CALL op1	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1
1	mem16 NEAR mem16
2	reg16 memi16
3	FAR mem16
4	memi16

Befehlswirkung:

Aufruf einer sich im Speicher befindlichen Prozedur (Unterprogramm).

Es werden 4 Typen des Befehls **CALL** unterschieden:

Typ 1: **Im Segment oder Gruppe, direkt**

Befehlsablauf: SP := SP - 2
[SP] := IP

Kodierung:

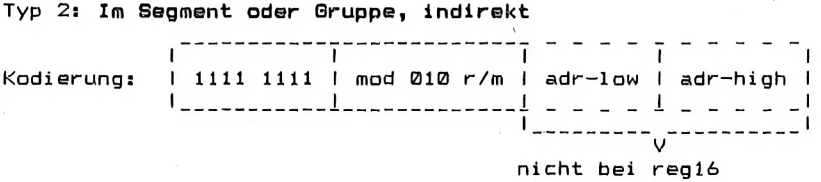
1110 1000	adr-low	adr-high

Beispiele:

```
CALL   NEAR UP1
CALL   UPROG
```

Es werden die Programme UP1 und UPROG im gleichen Kodesegment aufgerufen und abgearbeitet.

Typ 2: **Im Segment oder Gruppe, indirekt**



Befehlsablauf:

Wird ein 16-Bitregister (reg16) adressiert, dann wird der Inhalt dieses Registers in den Befehlszeiger (IP) uebertragen. Wird ein Speicherplatz (mem16) adressiert, so wird der Inhalt dieses und des darauffolgenden Speicherplatzes in den Befehlszeiger (IP) uebernommen.

Der Inhalt des Kodesegmentregisters (CS) wird nicht veraendert. Anschliessend wird der Befehl ausgefuehrt, dessen 1. Byte durch den neuen Inhalt des IP in Verbindung mit dem Inhalt des Registers CS adressiert wird.

Beispiele:

a) ; mit reg16

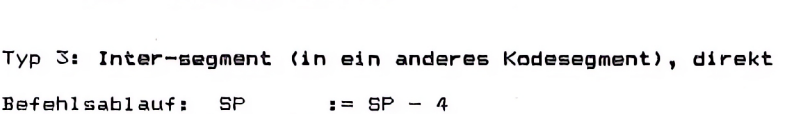
```
CALL   AX
CALL   DX
```

b) ; mit mem16

```
CALL   WORD PTR MARK[BX][SI]
CALL   WORD PTR [BP][DI]
```

Typ 3: **Inter-segment (in ein anderes Kodesegment), direkt**

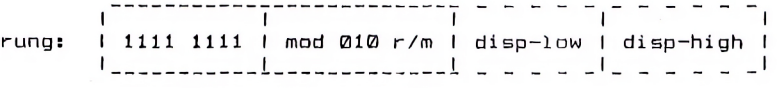
```
Befehlsablauf:  SP       := SP - 4
                [SP + 2] := CS
                [SP]     := IP
```



Beispiele:

```
CALL   FAR UP1
CALL   FAR UPROG
```

Typ 4: **Inter-segment, indirekt**



Operandenkombinationen: mem16

Befehlsablauf:

Der Inhalt des durch den Befehl adressierten Wortes wird in den Befehlszeiger (IP), der Inhalt des darauffolgenden Wortes wird in das Kodesegmentregister (CS) uebertragen. Anschliessend wird der Befehl ausgefuehrt, dessen absolute Adresse sich aus den neuen Inhalten von CS und IP ergibt.

Beispiele:

```
CALL   DWORD PTR PROG[BX][SI]
CALL   DWORD PTR [BX][DI]
```

4.11.2. Rueckkehr vom Unterprogramm

Befehl:	RET	Return from procedure
Schreibweise:	RET [op1]	

Flags: **keine Beeinflussung**

Operandenkombinationen: [imm16]

Befehlswirkung:

RET bewirkt die Rueckkehr vom einem mittels Befehl **Call** gerufenen Unterprogramms in das aufrufende Programm.

Es werden 4 Typen des Befehls **RET** unterschieden:

Typ 1: **Im gleichen Segment**

Befehlsablauf:
```
IP := [SP]
SP := SP + 2
```

```
             ------------
            |            |
Kodierung:  | 1100 0011  |
            |____________|
```

Beispiel:

```
    RET
```

Typ 2: **Aus einem anderen Kodesegment**

Befehlsablauf: IP := [SP]
CS := [SP + 2]
SP := SP + 4

```
             ------------
            |            |
Kodierung:  | 1100 1011  |
            |____________|
```

Beispiel:

```
    RET
```

Typ 3: **Im gleichen Segment mit Operand**

Befehlsablauf: IP := [SP]
SP := SP + 2 + imm16

```
             ------------------------------------
            |            |          |            |
Kodierung:  | 1100 0010  | imm-low  | imm-high   |
            |____________|__________|____________|
```

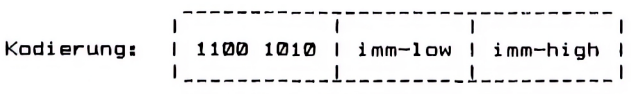
Beispiele:

```
    RET   4 ; Angabe in Bytes
    RET  12
```

Bedeutung: Diese Werte sagen aus, dass 2 und 6 Parameterworte einfach vom Stack genommen werden koennen, da die meisten Stackoperationen Wortoperationen sind und diese Werte eben als Zahlen genutzt werden (2 Bytes pro Wort), d.h., dass diese Parameter bei der Rueckkehr uebersprungen werden.

Typ 4: **Aus einem anderen Kodesegment mit Operand**

Befehlsablauf: IP := [SP]
CS := [SP + 2]
SP := SP + 4 + imm16



Beispiele:

```
RET   2
RET   8
```

4.11.3. Rueckkehr von der Unterbrechungsbehandlung

Befehl:	IRET	Return from interrupt
Schreibweise:	IRET	

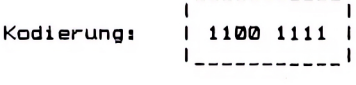
Flags: OF, DF, IF, TF, SF, ZF, AF, PF, CF

Operandenkombination: keine Operanden

Befehlswirkung:

```
SP     :=  SP + 6
[IP]    = [SP]
[CS]    = [SP - 4]
[Flags]:= [SP - 2]
```

IRET uebergibt die Steuerung an die Rueckkehradresse und holt die geretteten Register IP, CS und Flags aus dem Stack zurueck (wie POPF). Das Programm wird dann mit dem naechsten Befehl nach der Stelle fortgesetzt, an der der Interrupt auftrat.



4.12. CPU-Steuerbefehle

4.12.1. Coprozessor-Kommando mit Adresse

Befehl:	ESC	Escape
Schreibweise:	ESC op1, op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2
1	imm8	reg16
2	imm8	(mem16|memi16)

Befehlswirkung:

IF mod ≠ 11 THEN Datenbus := effektive Adresse 16 Bit
ELSE keine Operation
interner Coprozessorbefehl 0 ≤ imm8 ≤ 63

Der Befehl gibt ein Wort von einem Speicherplatz oder aus einem Register auf den Datenbus aus. Auf diese Art und Weise werden die K1810 WM86-Adressmodi und im Speicher befindliche Informationen anderen Prozessoren (Co-Prozessoren) zugaenglich gemacht.

Der 8086 Prozessor erledigt keine Operationen fuer den ESC-Befehl, ausser dem Zugriff auf einen Speicheroperanden und das Ablegen auf den Bus.

Beispiel:

```
ESC EXTERNAL_OPCODE,ADDRESS
```

4.12.2. Prozessorhalt

Befehl:	HLT	Halt
Schreibweise:	HLT	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:

Dieser Befehl bringt den Prozessor in den Halt-Status. Der Halt wird durch einen erlaubten externen Interrupt oder RESET aufgehoben.

Kodierung:

1111 0100

4.12.3. Synchronisation mit externer Hardware

Befehl:	WAIT	Wait
Schreibweise:	WAIT	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:

Der Befehl bringt den Prozessor in einen Wartezustand, solange kein RESET empfangen wird. Dieser Wartezustand kann durch einen erlaubten externen Interrupt unterbrochen werden. Dieser Interrupt wird ausgefuehrt und zum WAIT-Befehl zurueckgekehrt.

Das ermoeglicht dem Prozessor sich mit der externen Hardware zu synchronisieren.

Kodierung:

1001 1011

4.12.4. Prozessorinterrupt

Befehl:	INT	Interrupt
Schreibweise:	INT op1	

Flags: IF, TF : 0

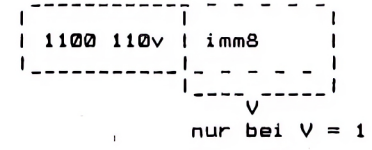
Operandenkombinationen: imm8

Befehlswirkung:

```
SP       := SP - 6
[SP + 4] := Flags
[SP + 2] := CS
[SP]     := IP
 IP      := [imm8 * 4]
 CS      := [imm8 * 4 + 2]
```

INT rettet die Flagregister (wie bei PUSHF), loescht die Flags TF und IF und uebergibt die Steuerung mit einem indirekten Aufruf an eine der 256 Adressen im Interruptvektor. Die Einbyteform dieses Befehls generiert einen Interrupt vom Typ 3 (INT 3).

Kodierung:



imm8 ≙ Interrupttyp

Beispiel:

```
INT 2       ; Interrupt 2  --> 2 Byte
INT 67      ; Interrupt 67 --> 2 Byte
INT 3       ; Interrupt 3  --> 1 Byte
```

4.12.5. Unterbrechung bei Ueberlauf

Befehl:	INTO	Interrupt if overflow
Schreibweise:	INTO	

Flags: IF, TF : 0

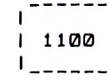
Operandenkombinationen: keine Operanden

Befehlswirkung: IF [OF] = 1 THEN INT 4

Falls das Ueberlaufflag OF gesetzt ist, rettet der Befehl das Flag-Register, loescht die Flags TF und IF und loest den Interrupt 4 (INT 4) aus.

Ist [OF] = 0, dann wird keine Operation ausgefuehrt.

Kodierung:



4.13. Ein-/Ausgabebefehle

4.13.1. Eingabe Byte oder Wort

Befehl:	IN	Input byte, word
Schreibweise:	IN op1, op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2	
1	AL	imm8	8-Bit-Portadresse
	AX	imm8	
2	AL	DX	16-Bit-Portadresse in DX
	AX	DX	

Befehlswirkung: [op1] := [op2]

IN transportiert ein Byte oder Wort von einem Eingabeport in das Register AL oder AX. Das Port wird spezifiziert durch einen Direktwert zwischen 0 und 255, oder dieser Direktwert steht im Register DX, wo 64 KByte Eingabeports moeglich sind. Wird AX als Zielregister verwendet, so wird ein Wort vom Eingabeport uebertragen, bei AL ein Byte.

Es werden 2 verschiedene Typen des Befehls unterschieden.

Typ 1: **Fester Port**

Kodierung:	1110 010w	imm8

Beispiele:

```
IN  AL,6    ; Der Inhalt der Portadresse 6  wird
            ; nach AL uebertragen (1 Byte).
IN  AX,23   ; Der Inhalt der Portadresse 23 wird
            ; nach AX uebertragen (1 Wort).
```

Typ 2: Variabler Port

```
                 ------------
                |            |
Kodierung:      | 1110 110w  |
                |____________|
```

Beispiele:

```
     ; Portadresse muss vorher in DX geladen werden

     IN  AX,DX    ; Wort
     IN  AL,DX    ; Byte
```

4.13.2. Ausgabe Byte oder Wort

Befehl:	OUT	**Output byte, word**
Schreibweise:	OUT op1, op2	

Flags: keine Beeinflussung

Operandenkombinationen:

Typ	op1	op2	
1	imm8	AL	
	imm8	AX	8-Bit-Portadresse
2	DX	AL	
	DX	AX	16-Bit-Portadresse

Befehlswirkung: [op1] := [op2]

OUT transportiert ein Byte oder ein Wort vom Register AL oder AX zu einem Ausgabeport. Die Portadresse muss ein Direktwert zwischen 0 und 255 sein. Steht die Portadresse im Register DX, dann koennen 64 K Ports adressiert werden.

Es werden 2 verschiedene Typen des Befehls unterschieden:

Typ 1: **Fester Port**

```
                 ------------------------
                |            |           |
Kodierung:      | 1110 011w  |  imm8     |
                |____________|___________|
```

Beispiele:

```
OUT   44,AL  ; Byte
OUT  125,AX  ; Wort
```

Typ 2: **Variabler Port**

```
                 -----------
                |           |
Kodierung:      | 1110 111w |
                |-----------|
```

Beispiele:

```
OUT  DX,AL  ; Byte
OUT  DX,AX  ; Wort
```

4.14. Korrekturbefehle

4.14.1. ASCII-Korrektur nach Addition

Befehl:	AAA	ASCII adjust fuer addition
Schreibweise:	AAA	

Flags: AF, CF : X
OF, SF, ZF, PF: U

Operandekombinationen: keine Operanden

Befehlswirkung: IF ([AL] & 0FH) > 9 | [AF] = 1

```
THEN
     [AL] := [AL] + 6
     [AH] := [AH] + 1

Flags: [AF] := 1
       [CF] := [AF]

       [AL] := [AL] & 0FH
```

AAA fuehrt eine Korrektur des Resultates im Register AL nach einer Addition zweier ungepackter Operanden durch. Dabei entsteht eine ungepackte Summe.

Sind die unteren 4 Bits von AL groesser als 9 oder das Flag AF ist gesetzt, dann wird eine 6 zu AL addiert und AH um 1 erhoeht. AF und CF werden gesetzt.

Der neue Wert in AL hat in den oberen 4 Bits Nullen und die unteren 4 Bits enthalten die Zahl zwischen 0 und 9.

```
                 ------------
                |            |
Kodierung:      | 0011 0111  |
                |____________|
```

Um einen ASCII-Wert wieder zu erreichen muss der Befehl OR AX,3030H anschliessend gegeben werden.

Beispiel:

```
     MOV   AH,0
     ADD   AL,DL
     AAA
     OR    AX,3030H
```

In AL soll 36H und in DL 38H vor der Addition stehen. Der Befehl berechnet folgende Summe:

```
       0011 0110  : 36H
   +   0011 1000  : 38H
   --------------------
       0110 1110  : 6EH
```

In AL steht nach der Addition 6EH. Nach dem Befehl AAA steht dann in AH : 01 und in AL : 04.

Nach Ausfuehrung von OR AX,3030H steht die Summe wieder ASCII-gerecht in AX : 3134H.

4.14.2. ASCII-Korrektur vor Division

Befehl:	AAD	ASCII adjust for division
Schreibweise:	AAD	

Flags: SF, ZF, PF: X
OF, AF, CF: U

Operandenkombinationen: keine Operanden

Befehlswirkung: [AL] = [AH] * 0AH + [AL]
[AH] := 0

AAD wird zur Vorbereitung des Dividenden in AL angewandt. Der Befehl muss vor dem Divisionsbefehl geschrieben werden.

Resultat ist ein ungepackter Dividend.

Das Byte in AH wird mit 10 multipliziert und zu AL addiert. Das Resultat wird in AL abgespeichert. AH wird Null.

Kodierung:	1101 0101	0000 1010

4.14.3. ASCII-Korrektur nach Multiplikation

Befehl:	AAM	ASCII adjust for multiply
Schreibweise:	AAM	

Flags: SF, ZF, PF: X
OF, AF, CF: U

Operandenkombinationen: keine Operanden

Befehlswirkung: [AH] := [AL] / 0AH
[AL] := [AL] mod 0AH

AAM fuehrt eine Korrektur des Resultates in AX durch. Ergebnis ist ein ungepacktes dezimales Produkt.

Der Inhalt von AH wird durch 10 dividiert. Anschliessend wird der Inhalt von AL durch den Divisionrest ersetzt.

Kodierung:	1101 0100	0000 1010

4.14.4. ASCII-Korrektur nach Subtraktion

Befehl:	AAS	ASCII adjust for subtraction
Schreibweise:	AAS	

Flags: AF, CF : X
OF, ZF, SF, PF: U

Operandenkombinationen: keine Operanden

Befehlswirkung: IF ([AL] & 0FH) > 9 oder [AF] = 1
THEN

```
[AL] := [AL] - 6
[AH] := [AH] - 1
[AL] := [AL] & 0FH
```

Flags: [AF] := 1
[CF] := 1

AAS fuehrt eine Korrektur des Resultates nach einer Subtraktion zweier ungepackter Zahlen durch. Das Ergebnis ist eine ungepackte Differenz.

Sind die niederen 4 Bits von AL groesser als 9, oder das Register AF ist gesetzt, dann wird eine 6 von AL subtrahiert und AH um 1 vermindert. Die Flags AF und CF werden gesetzt. In AL entsteht ein Byte, dessen hoeherwertiger Teil 0 und dessen niederwertiger Teil einen Wert zwischen 0 und 9 belegt.

Mit dem anschliessenden Befehl OR AX,3030H wird eine ASCII-Zahl erreicht.

Kodierung	0011 1111

4.14.5. Dezimalkorrektur nach Addition

Befehl:	**DAA**	**Dezimal adjust for addition**
Schreibweise:	**DAA**	

Flags: SF, ZF, AF, PF, CF: X
OF : U

Operandenkombinationen: keine Operanden

Befehlswirkung:

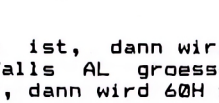
```
IF ([AL] & 0FH) > 9  | [AF] = 1
   THEN [AL] := [AL] + 6
        [AF] := 1

IF  [AL] > 9FH       | [CF] = 1
   THEN [AL] := [AL] + 60H
        [CF] := 1
```

DAA fuehrt eine Korrektur des Resultates nach der Addition zweier gepackter Zahlen in AL durch, um eine gepackte dezimale Summe zu erhalten.

Sind die unteren 4 Bits des Registers AL groesser als 9 oder ist

AF gesetzt ist, dann wird eine 6 zu AL addiert und AF wird gesetzt. Falls AL groesser als 9FH oder das Uebertragsflag gesetzt ist, dann wird 60H zu AL addiert und CF gesetzt.



4.14.6. Dezimalkorrektur nach Subtraktion

Befehl:	DAS	Decimal adjust for subtraction
Schreibweise:	DAS	

Flags: SF, ZF, AF, PF, CF: X
OF : U

Operandenkombinationen: keine Operanden

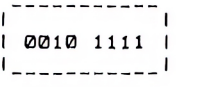
Befehlswirkung:
```
IF ([AL] & 0FH) > 9 | [AF] = 1
   THEN [AL] := [AL] - 6
        [AF] := 1

IF [AL] > 9FH | [CF] = 1
   THEN [AL] := [AL] - 60H
        [CF] := 1
```

DAS fuehrt eine Korrektur des Resultates nach der Subtraktion zweier gepackter Zahlen in AL durch, um eine gepackte dezimale Differenz zu erhalten.

Zuerst wird die niederwertige Stelle behandelt. Sind die niederwertigen 4 Bit von AL groesser als 9 oder ist AF gesetzt, wird von AL der Wert 6 subtrahiert und das Flag AF gesetzt.

Danach wird der Zustand der zweiten Dezimalstelle kontrolliert. Ist das Flag CF gesetzt oder AL groesser als 9FH, wird 60H von AL subtrahiert und CF gesetzt.



4.15. Wandelbefehle

4.15.1. Umwandlung Byte in Wort

Befehl:	CBW	Convert byte to word
Schreibweise:	CBW	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung: IF [AL] < 80H THEN [AH] := 0
ELSE [AH] := 0FFH

CBW wandelt das in AL stehende Byte in ein in AX stehendes Wort. Wenn das niederwertige Byte von AL kleiner als 80H ist, dann wird AH zu 0. Sonst wird AH auf 0FFH gesetzt, d.h., Bit 7 von AL fuellt die Bits von AH.

Kodierung:

1001 1000

4.15.2. Umwandlung Wort in Doppelwort

Befehl:	CWD	Convert word to doubleword
Schreibweise:	CWD	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung: IF [AX] < 8000H THEN [DX] := 0
ELSE [DX] := 0FFFFH

CWD wandelt ein in AX stehendes Wort in ein Doppelwort um, dass in den Registern DX und AX steht.

Ist der Wert in AX kleiner 8000H, dann wird DX mit 0 gefuellt. Das hoechste Bit (Bit 15) von AX fuellt das Register DX.

Kodierung:

1001 1001

4.16.Zeichenkettenbefehle

4.16.1. Vergleich Bytekette/Wortkette

Befehl:	CMPS fuer beide CMPSW Wortkette CMPSB Bytekette	Compare word string Compare byte string
Schreibweise:	CMPS CMPSW CMPSB	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen: keine Operanden

Befehlswirkung: Vergleich [DS:SI] mit [ES:DI]

```
IF [DF] = 0
   THEN [SI] := [SI] + DELTA
        [DI] := [DI] + DELTA
   ELSE
        [SI] := [SI] - DELTA
        [DI] := [DI] - DELTA
```

Dabei ist DELTA = 1 fuer Byteketten und
DELTA = 2 fuer Wortketten.

Vor dem Befehl muessen die Register SI und DI mit den Adressen der zu vergleichenden Ketten geladen werden.

Diese Befehle vergleichen das durch SI adressierte Byte oder Wort im Datensegment mit dem durch DI adressierten Byte oder Wort im Extrasegment. Dabei werden die Flags entsprechend eingestellt. SI und DI werden bei DF = 1 inkrementiert. Der Inkrement- bzw. Dekrementwert ist fuer Byteketten 1 und fuer Wortketten 2. Nach der Operation zeigen dann SI und DI auf das naechste Element der zu vergleichenden Ketten. Diese Operation kann mit den REP-Praefixen mehrfach ausgefuehrt werden.

Kodierung:

1010 011w

Beispiel:

```
MOV   SI,OFFSET KETTE1
MOV   DI,OFFSET KETTE2
CMPSW
```

4.16.2. Laden Byte- oder Wortkette in den Akkumulator

Befehl:	LODS fuer beide	
	LODSW Wortkette	Load word string
	LODSB Bytekette	Load byte string
Schreibweise:	LODS	
	LODSW	
	LODSB	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:
```
[AL | AX] := [SI]
IF [DF] = 0
   THEN [SI] := [SI] + DELTA
   ELSE [SI] := [SI] - DELTA

DELTA = 1 fuer Byteketten
DELTA = 2 fuer Wortketten
```

LODS transportiert ein Byte oder Wort vom durch das Register SI (bezueglich Datensegment DS) adressierten Quellfeld zum Akkumulator AL oder AX und veraendert das Register SI um DELTA. Mit den REP-Praefixen kann die Operation mehrfach ausgefuehrt werden.

Kodierung:

1010 110w

Beispiel:

```
      CLD          ;Loeschen DF --> Erhoehen von SI
      MOV   SI,OFFSET KETTEB
      LODSB        ;Bytekette [SI] := [SI] + 1
        .
        .
        .
      STD          ;Setzen DF --> SI vermindern
      MOV   SI,OFFSET KETTEW
      LODSW        ;Wortkette [SI] := [SI] - 2
```

4.16.3. Blocktransport im Speicher

Befehl:	MOVS fuer beide MOVSW fuer Wort MOVSB fuer Byte	Move word string Move byte string
Schreibweise:	MOVS MOVSW MOVSB	

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:
```
[ziel] := [quelle]
IF [DF] = 0
   THEN [SI] := [SI] + DELTA
        [DI] := [DI] + DELTA

ELSE
   [SI] := [SI] - DELTA
   [DI] := [DI] - DELTA

DELTA = 1 fuer Byteketten
DELTA = 2 fuer Wortketten
```

MOVS transportiert ein Byte oder Wort vom durch SI adressierten Quellfeld im Datensegment zum durch DI adressierten Zielfeld im Extrasegment. Anschliessend werden SI und DI um DELTA entsprechend [DF] de- bzw. inkrementiert. Mit Anwendung des Praefix REP kann die Operation mehrfach ausgefuehrt werden.

Kodierung: | 1010 010w |

Beispiel:

```
        MOV     SI,OFFSET QUELLE
        MOV     DI,OFFSET ZIEL
        MOV     CX,LAENGE_QUELLE
REP     MOVSB
```

Die Anweisung mit REP MOVSB uebertraegt die gesamte Quellkette aus dem Datensegment in die Zielposition im Extrasegment. Ohne REP wuerde nur 1 Byte bei obigem Beispiel in den Zielbereich uebertragen.

4.16.4. Suchen Akkumulatorinhalt im Speicher

Befehl:	SCAS fuer beide SCASW fuer Wort SCASB fuer Byte	 Scan word string Scan byte string
Schreibweise:	SCAS SCASW SCASB	

Flags: OF, SF, ZF, AF, PF, CF: X

Operandenkombinationen: keine Operanden

Befehlswirkung: Vergleich [AL | AX] mit [mem8 | mem16]
IF [DF] = 0
THEN [DI] := [DI] + DELTA
ELSE [DI] := [DI] - DELTA

DELTA = 1 fuer Byteketten
DELTA = 2 fuer Wortketten

SCAS subtrahiert das durch DI im Extrasegment adressierte Byte oder Wort im Speicher vom AL- oder AX-Inhalt und setzt die Flags. Das Register und der Speicheroperand bleiben dabei erhalten. Das Indexregister DI wird entsprechend [DF] in- bzw. dekrementiert. Die Operation kann mit Hilfe der REP-Praefixe mehrfach ausgefuehrt werden.

Kodierung:

1010 111w

Beispiel:

```
CLD          ;Loeschen Richtungsflag DF
MOV   DI,OFFSET KETTE
MOV   AL,'M'; zu suchender Begriff
SCASB
```

Dieser Befehl SCASB vergleicht den in AL stehenden Buchstaben 'M' mit dem 1. Byte der Kette und erhoeht DI anschliessend um 1. Anhand der Flageinstellung ist zu erkennen, ob das Byte der Kette, durch DI adressiert, mit dem Wert in AL uebereinstimmt.

4.16.5. Speicher mit Akkumulatorinhalt laden

Befehl:	STOS fuer beide		
	STOSW fuer Wort	Store word string	
	STOSB fuer Byte	Store byte string	
Schreibweise:	STOS		
	STOSW		
	STOSB		

Flags: keine Beeinflussung

Operandenkombinationen: keine Operanden

Befehlswirkung:
```
[mem8 | mem16] := [AL | AX]
IF [DF] = 0
   THEN [DI] := [DI] + DELTA
   ELSE [DI] := [DI] - DELTA

DELTA = 1 fuer Byteketten
DELTA = 2 fuer Wortketten
```

Das Byte oder Wort in AL oder AX ersetzt den Inhalt des Bytes oder Wortes im Extrasegment, das durch DI adressiert wird. DI wird anschliessend entsprechend [DF] in- bzw. dekrementiert. Mit Verwendung von REP-Praefixen kann die Operation mehrfach ausgefuehrt werden.

Kodierung:	1010 101w

Beispiel:

```
CLD          ;Loeschen DF
MOV   DI,OFFSET KETTE
MOV   AX,'AB'
STOSW
```

Der Befehl STOSW schreibt das in AX stehende Wort 'AB' auf den Speicherplatz KETTE, der durch DI adressiert ist und erhoeht DI anschliessend um 2.

Anhang A

Zusammenstellung aller Befehle, alphabetisch geordnet

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
ASCII-Korrektur nach Addition	AAA	-	1	U...UUXUX
ASCII-Korrektur vor Division	AAD	-	2	U...XXUXU
ASCII-Korrektur nach Multiplikation	AAM	-	1	U...XXUXU
ASCII-Korrektur nach Subtraktion	AAS	-	1	U...UUXUX
Addition mit Uebertrag	ADC	reg8 ,reg8	2	X...XXXXX
		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Addition ohne Uebertrag	ADD	reg8 ,re8	2	X...XXXXX
		reg16 ,reg16	2	
		(mem8|memi8 ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Logisches UND	**AND**	reg8 ,reg8	2	0...XXUX0
		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Unterprogramm-aufruf	**CALL**	mem16	3	.....
		NEAR mem16	3	
		reg16	2 - 3	
		memi16	2 - 4	
		FAR mem16	5	
		memi16	4	
Umwandlung Byte in in Wort	**CBW**	-	1	.....
Uebertragsflag rueckssetzen	**CLC**	-	1	0
Richtungsflag rueckssetzen	**CLD**	-	1	.0.....
Interruptflag rueckssetzen	**CLI**	-	1	..0.....
Uebertragsflag negieren	**CMC**	-	1	X

Bezeichnung	Befehl	Operanden		Bytes	Flags ODITSZAPC
Vergleich	**CMP**	reg8	,reg8	2	X...XXXXX
		reg16	,reg16	2	
		reg8	,(mem8|memi8)	2 - 4	
		(mem8|memi8)	,reg8	2 - 4	
		reg16	,(mem16|memi16)	2 - 4	
		(mem16|memi16)	,reg16	2 - 4	
		AL	,imm8	2	
		AX	,imm16	3	
		reg8	,imm8	2 - 3	
		reg16	,imm16	3 - 4	
		(mem8|memi8)	,imm8	3 - 5	
		(mem16|memi16)	,imm16	4 - 6	
Vergleich Bytekette / Wortkette	**CMPS** **CMPSB** **CMPSW**		-	1	X...XXXXX
Umwandlung Wort in Doppelwort	**CWD**		-	1	.....
Dezimalkorrektur nach Addition	**DAA**		-	1	U...XXXXX
Dezimalkorrektur nach Subtraktion	**DAS**		-	1	U...XXXXX
Dekrementieren	**DEC**	reg16		1	X...XXXX.
		reg8		2	
		mem8		4	
		mem16		4	
		memi8		2 - 4	
		memi16		2 - 4	
Division vorzeichenlos	**DIV**	reg8		2	U...UUUUU
		reg16		2	
		mem8		4	
		mem16		4	
		memi8		2 - 4	
		memi16		2 - 4	
Koprozessorkommando mit Adresse	**ESC**	imm8	,reg16	2	.....
		imm8	,(mem16|memi16)	2 - 4	
Prozessorhalt	**HLT**		-	1	.....

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Division vorzeichenbehaftet, ganzzahlig	IDIV	reg8	2	U...UUUUU
		reg16	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Multiplikation vorzeichenbehaftet, ganzzahlig	IMUL	reg8	2	X...UUUUX
		reg16	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Eingabe Byte oder Wort	IN	AL ,imm8	2	.....
		AX ,imm8	2	
		AL ,DX	1	
		AX ,DX	1	
Inkrementieren	INC	reg16	1	X...XXXX.
		reg8	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Prozessorinterrupt	INT	imm8	2	..00.....
		3	1	
Unterbrechung bei Ueberlauf	INTO	-	1	..00.....
Rueckkehr von der Unterbrechungs-behandlung	IRET	-	1	XXXXXXXXX
Bedingte Spruenge mit kurzer Distanz -128 ≤ sdisp ≤ +127 (siehe Tabelle im Abschnitt 4.8.1., CPU-Befehlsbe-schreibung)	Jcond	sdisp	2	.....

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Unbedingter Sprung	JMP	mem16	3	.....
		SHORT mem16	2	
		reg16	2 - 3	
		mem16	2 - 4	
		FAR mem16	5	
		mem16	4	
Flags nach AH laden	LAHF	-	1	.....
Laden Datensegment-register	LDS	reg16 ,mem16	4	.....
		reg16 ,mem16	2 - 4	
Laden effektive Adresse	LEA	reg16 ,mem16	4	.....
		reg16 ,mem16	2 - 4	
Laden Extrasegment register	LES	reg16 ,mem16	4	.....
		reg16 ,mem16	2 - 4	
Laden Byte- oder Wortkette in den Akkumulator	LODS LODSB LODSW	-	1	.....
Schleifenbefehl	LOOP	sdisp	2	.....
Bedingte Schleifenbefehle mit Spring bei Gleich oder Null	LOOPE LOOPZ	sdisp	2	.....
Bedingte Schleifenbefehle mit Sprung bei Ungleich oder nicht Null	LOOPNE LOOPNZ	sdisp	2	.....

*** CPU - BEFEHLSLISTE ***

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Uebertragen von Daten	**MOV**	memb16 ,AX	2 - 4	.....
		memb8 ,AL	2 - 4	
		AX ,memb16	2 - 4	
		AL ,memb8	2 - 4	
		sreg(auseer CS),reg16	2	
		sreg(ausser CS),(mem16|memi16)	2 - 4	
		reg16 ,sreg	2	
		(mem16|memi16),sreg	2 - 4	
		reg8 ,reg8	2	
		reg16 ,reg16	2	
		reg8 ,(mem8|memi8)	2 - 4	
		(mem8|memi8) ,reg8	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,imm8	2	
		reg16 ,imm16	3	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Blocktransport im Speicher Byte Wort	**MOVS** **MOVSB** **MOVSW**	-	1	.....
Multiplikation vorzeichenlos	**MUL**	reg8	2	X...UUUUX
		reg16	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Arithmetische Negation - Zweierkomplement	**NEG**	reg8	2	X...XXXXX
		reg16	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Leeroperation	**NOP**	-	1	.....

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Logisches Komplement	NOT	reg8	2	.....
		reg16	2	
		mem8	4	
		mem16	4	
		memi8	2 - 4	
		memi16	2 - 4	
Logisches ODER	OR	reg8 ,reg8	2	0...XXUX0
		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Ausgabe Byte oder Wort	OUT	imm8 ,AL	2	.....
		imm8 ,AX	2	
		DX ,AL	1	
		DX ,AX	1	
Stack leeren	POP	reg16	1	.....
		sreg (ausser CS!)	1	
		mem16	4	
		memi16		
Flags aus dem Stack laden	POPF	-	1	XXXXXXXXX
Stack fuellen	PUSH	reg16	1	.....
		sreg	1	
		mem16	4	
		memi16	2 - 4	
Flags in den Stack laden	PUSHF	-	1	.....

*** CPU - BEFEHLSLISTE ***

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Linksrotation durch CF	RCL	reg8 ,1	2	X.....X
		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	
Rechtsrotation durch CF	RCR	reg8 ,1	2	X.....X
		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	
Rueckkehr vom Unterprogramm	RET	-	1	.....
		imm16	3	
Linksrotation	ROL	reg8 ,1	2	X.....X
		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	
Rechtsrotation	ROR	reg8 ,1	2	X.....X
		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL		
		(mem16|memi16),CL		
Flags von AH in das Flagregister laden	SAHF	-	1	XXXXX

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Arithmetische oder	SAL	reg8 ,1	2	X...XXUXX
logische	SHL	reg16 ,1	2	
Linksverschiebung		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	
Arithmetische	SAR	reg8 ,1	2	X...XXUXX
Rechtsverschiebung		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	
Subtraktion mit	SBB	reg8 ,reg8	2	X...XXXXX
Uebertrag		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Suchen Akkumulator-	SCAS	-	1	X...XXXXX
inhalt im Speicher				
Byte	SCASB			
Wort	SCASW			
Logische	SHR	reg8 ,1	2	X...XXUXX
Rechtsverschiebung		reg16 ,1	2	
		(mem8|memi8) ,1	2 - 4	
		(mem16|memi16),1	2 - 4	
		reg8 ,CL	2	
		reg16 ,CL	2	
		(mem8|memi8) ,CL	2 - 4	
		(mem16|memi16),CL	2 - 4	

Bezeichnung	Befehl	Operanden	Bytes	Flags ODITSZAPC
Uebertragsflag setzen	STC	-	1	1
Richtungsflag setzen	STD	-	1	.1.....
Interruptflag setzen	STI	-	1	..1.....
Speicher mit Akkumulatorinhalt laden	STOS	-	1	.....
Byte	STOSB			
Wort	STOSW			
Subtraktion ohne Uebertrag	SUB	reg8 ,reg8	2	X...XXXXX
		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16|	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16 ,imm16	4 - 6	
Logisches UND ohne Resultatsabspeicherung	TEST	reg8 ,reg8	2	0...XXUX0
		reg16 ,reg16	2	
		(mem8|memi8) ,reg8	2 - 4	
		(mem16|memi16),reg16	2 - 4	
		reg8 ,(mem8|memi8)	2 - 4	
		reg16 ,(mem16|memi16)	2 - 4	
		AL ,imm8	2	
		AX ,imm16	3	
		reg8 ,imm8	2 - 3	
		reg16 ,imm16	3 - 4	
		(mem8|memi8) ,imm8	3 - 5	
		(mem16|memi16),imm16	4 - 6	
Synchronisation mit externer Hardware	WAIT	-	1	.....

Bezeichnung	Befehl	Operanden		Bytes	Flags ODITSZAPC
Austauschbefehl	XCHG	AX	,reg16	1	.....
		reg16	,reg16	1 - 2	
		reg8	,reg8	2	
		(mem8|memi8)	,reg8	2 - 4	
		(mem16|memi16)	,reg16	2 - 4	
Umkodierung von AL	XLAT XLATB	-		1	.....
Exklusives ODER	XOR	reg8	,reg8	2	0...XXUX0
		reg16	,reg16	2	
		(mem8|memi8)	,reg8	2 - 4	
		(mem16|memi16)	,reg16	2 - 4	
		reg8	,(mem8|memi8)	2 - 4	
		reg16	,(mem16|memi16)	2 - 4	
		AL	,imm8	2	
		AX	,imm16	3	
		reg8	,imm8	2 - 3	
		reg16	,imm16	3 - 4	
		(mem8|memi8)	,imm8	3 - 5	
		(mem16|memi16)	,imm16	4 - 6	

Beachte:

Bei Verwendung eines Segmentpraefixes fuer einen Operanden erhoeht sich die angegebene Byteanzahl um 1. Es wird ein Vorbyte generiert.

Praefixe

Sie werden vor dem eigentlichen Befehl geschrieben und liefern eine Vorbyte.

Bedeutung	Praefix	Vorbyte
Bus-Lock-Signal fuer folgenden Befehl aktivieren	LOCK	F0
Wiederholungspraefixe fuer LODS, MOVS, STOS	REP	F2
Wiederholungspraefixe fuer CMPS und SCAS	REPE	F3
	REPNE	F2
	REPNZ	F2
	REPZ	F3

Abkuerzungen

Die Abkuerzungen entsprechen der Metasprache im Abschnitt 2 der CPU-Befehlsbeschreibung. Eine Ausnahme bildet die Bezeichnung der Flags. Aus Platzgruenden wurde in dieser Befehlsliste nur der erste Buchstabe verwendet.

1. 62. 548308. 5. D